MASTER EN SCIENCES INFORMATIQUES

Analyse de la couche session du modèle de référence ISO

Valentin, Didier

Award date:
1988

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix à Namur
Institut d'Informatique

ANALYSE DE LA COUCHE SESSION DU MODELE DE REFERENCE ISO

Mémoire présenté par Didier Valentin
en vue de l'obtention du titre de
Licencié et Maître en Informatique

Promoteur : P. Van Bastelaer

Année Académique 1987-1988

Je tiens à remercier Monsieur P. Van Bastelaer, promoteur de ce mémoire, pour avoir accepté de diriger ce travail et pour ses critiques lors de la rédaction de ce manuscrit.

J'exprime toute ma gratitude à Monsieur S. Simonet pour l'intérêt qu'il a porté à ce mémoire, ses critiques et ses réflexions, ainsi que pour l'attention toute particulière qu'il a manifestée à mon égard.

Je tiens à exprimer, par ailleurs, ma reconnaissance à tous ceux qui de près ou de loin ont contribué à la réalisation de ce travail.

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ABREVIATIONS

ACS : Association Control Service
APDU : Application Protocol Data Unit
BAS : Basic Activity Subset
BCS : Basic Combined Subset
BSS : Basic Synchronized Subset
CASE : Common Application Service Element
CCITT : Comité Consultatif International des Télégraphes et des Téléphones
CCR : Commitment Concurrency and Recovery
CDCL : Command Document Capability List
CEP : Connection End Point
DoD : Department of Defense
ECMA : European Computer Manufacturer Association
FTAM : File Transfer and Access Management
IS : International Standard
ISO : International Standards Organization
ISO IS 8326 : norme concernant le service session
ISO IS 8327 : norme concernant le protocole session
ISODE : programme implémentant les couches supérieures de l'ISO
IP : Internet Protocol
JTM : Job Transfer and Management
MHS : Message Handling System
OS : Operating System
OSI : Open System Interconnection
PDU : Protocol Data Unit
PPDU : Presentation Protocol Data Unit
QOS : Quality Of Service
RDCL : Response Document Capability List
ROS : Remote Operation Service
RTS : Reliable Transfer Service
SAP : Service Access Point
SDU : Service Data Unit
SPDU : Session Protocol Data Unit
SPM : Session Protocol Machine
SSAP : Session Service Access Point
SSDU : Session Service Data Unit

T 62	: norme concernant le télétex
TCP	: Transmission Control Protocol
TSDU	: Transport Service Data Unit
VT	: Virtual Terminal
VTS	: Virtual Terminal Service
X 215	: norme concernant le service session
X 225	: norme concernant le protocole session
X 400	: norme concernant le MHS (System Model and Services Elements)
X 410	: norme concernant le RTS et le ROS du MHS

I INTRODUCTION

Dans les revues d'informatique mais surtout de téléinformatique et de télécommunication, les auteurs parlent beaucoup du modèle de référence à 7 couches de l'ISO (physique, liaison de données, réseau, transport, session, présentation, application). Parmi ces dernières, la session apparaît comme un parent pauvre. Peut être cela provient-il de la définition très abstraite des services offerts par cette couche où l'on retrouve des termes tels que organisation, synchronisation ? Le fait est que les auteurs en parlent peu ou très vaguement, n'insistent pas sur les mêmes points et même, de temps en temps, se trouvent en contradiction dans leurs diverses interprétations. Dans l'espoir de lever une partie du voile qui repose sur ce niveau, nous en avons entrepris une analyse approfondie.

Cette analyse, nous l'avons voulue la plus complète possible, dans la limite du temps qui nous était imparti. Nous avons surtout porté notre intérêt sur la théorie : nous avons étudié les normes qui concernaient directement la session ou qui en utilisaient les services. Toutefois, nous n'avons pas délaissé le côté pratique : nous avons installé et analysé une implémentation de la session et des autres couches supérieures du modèle de référence (présentation, application).

Ce mémoire se décompose en deux parties : d'une part, une étude de la normalisation et d'autre part, une étude d'une implémentation. Notre étude de la normalisation comporte deux chapitres. Le premier est une introduction générale à la session. Son objectif est de donner au lecteur une bonne idée du type de couche session que nous allons analyser et des services qu'elle offre. Pour cela, nous ferons le point sur la normalisation en la matière. Ensuite, nous présenterons les différents concepts reliés à la session et nous tenterons de justifier leur présence à ce niveau du modèle de référence. Le deuxième chapitre est une analyse approfondie des services de ce niveau et de l'utilisation que l'on peut en faire. L'objectif est ici de déterminer ce que fait réellement la session pour arriver à déterminer des critères d'utilisation de ses services. Pour cela, nous présenterons notre analyse ainsi qu'une démarche personnelle permettant de lier services offerts et besoins d'un programme d'application.

La deuxième partie de notre travail est consacrée à l'étude d'une implémentation de la session (ISODE). L'objectif est ici de découvrir les problèmes liés à la pratique. Cette partie n'est composée que d'un seul chapitre. Il comprend une présentation et une analyse générales du logiciel : explication de l'architecture, analyse de la session (complétude, conformité), présentation des problèmes liés à l'interface, l'adressage, l'intialisation des couches. De plus, afin d'illustrer l'ensemble de nos propos, nous présenterons une petite application personnelle de transfert de fichiers utilisant directement les services session.

Nous espérons vivement que notre travail pourra faciliter la compréhension de cette couche et aider à une utilisation correcte de ses services. Nous sommes toutefois conscient de sa lourdeur. Elle est dûe principalement à deux éléments : l'abstraction des concepts manipulés et le nombre des services offerts par la session. Nous espérons que notre présentation, de par ses nombreux exemples, schémas, regroupements et comparaisons, permettra de l'alléger quelque peu.

II CHAPITRE 1 : PRESENTATION GENERALE DE LA COUCHE

1 Introduction

L'objectif de ce chapitre est d'introduire de façon générale la couche session ISO. Il permet de répondre à certaines questions. Quelles sont les normes qui définissent la session ? Y a-t-il des liens avec d'autres normalisations ? Quelles sont les concepts qui se cachent derrière l'expression "couche session" ? L'existence de cette couche se justifie-t-elle ?

Pour ce faire, nous présenterons, en un premier lieu, l'état de la normalisation ISO et d'autres normalisations en matière de session, ainsi que leurs interactions. Ensuite, nous donnerons une définition claire et précise de cette couche. Ceci nous permettra d'aborder, dans un troisième volet, les différents concepts mis en jeu par la session. Finalement, nous tenterons de justifier l'existence de ce niveau. Nous montrerons pour cela, en un premier temps, l'utilité de la session et exposerons quelques exemples d'utilisation. Ensuite, nous tenterons de justifier le point de vue pris par l'ISO à propos des concepts présentés et de leur niveau dans l'architecture.

2 Etat de la normalisation

La couche session occupe le niveau 5 tel que défini dans l'architecture ISO concrétisée par la norme ISO 7498 "Système de traitement de l'information - Interconnexion des systèmes ouverts - Modèle de référence."

Le service offert par la couche session est défini dans la norme ISO IS 8326. Il est orienté connexion (voir section suivante). Il est équivalent à la recommandation X215 du CCITT. II existe deux addenda concernant ce service. Le premier est une définition du service session couvrant le mode de transmission orienté non-connexion. Le second définit un service supplémentaire de synchronisation symétrique. Ces deux addenda ne sont pas encore devenus des standards internationaux.

La spécification du protocole session orienté connexion est définie dans la norme ISO IS 8327. Elle est équivalente à la recommandation X225 du CCITT. La description du protocole repose sur la définition du service transport ISO IS 8072. La norme possède un addendum concernant la synchronisation symétrique. Cet addendum, ainsi que la spécification d'un protocole session orienté non-connexion, n'ont pas encore été concrétisés en standards internationaux par l'ISO.

Les concepteurs ISO se sont basés sur le standard ECMA-75 pour réaliser les normes 8326 et 8327. Ils y ont intégré par la suite le protocole T62 du CCITT. C'est pour cette raison que, dans leurs formes actuelles, on y retrouve des traces de ces standards.

3 Portée de notre étude

Notre travail porte essentiellement sur une session orientée connexion. Rappelons les caractéristiques de ce mode de transmission :

- "la décomposition de la communication en trois phases distinctes (ouverture de la connexion, transfert des données, clôture)
- la négociation des caractéristiques de l'échange entre trois parties (le fournisseur et les deux utilisateurs)
- l'utilisation d'un identificateur de connexion
- la relation de séquence entre les données échangées sur une même connexion." [CHA 82]

Par la suite nous ferons peu référence au mode de transmission orienté non-connexion. Rappelons que ce mode correspond à "un échange d'unité de données indépendantes, appelées aussi datagrammes." [CHA 83]

Notre choix a été influencé par deux éléments. Premièrement, il n'y a pas de standard international de niveau session concernant la transmission de données orientée non-connexion. Les concepteurs de l'ISO y travaillent actuellement pour garder une portée générale au modèle de référence. Deuxièmement, la session ISO est, comme nous le verrons par la suite, de nature orientée connexion. Comme le signale Lyman Chapin dans un article sur les modes de transmission, "l'introduction d'un service et d'un protocole de niveau 5 orientés non-connexion a pour seul objectif de permettre une réalisation efficace de ce mode de transmission à travers l'ensemble des couches." [CHA 83]

Les addenda sur la synchronisation symétrique furent introduits pour pallier une déficience des normes en matière de communication full duplex. Nous ne développerons pas ce mécanisme par la suite. Nous en soulignerons simplement l'utilité.

4 Définition du service session

Rappelons que la couche session repose sur le service transport. Ce service de niveau 4 permet l'échange full duplex de données sur une connexion de bout en bout, à un niveau de qualité désiré, quel que soit le réseau utilisé.

Imaginons un transfert de fichiers entre deux applications reposant sur le service transport. Comment peut-on réagir lors d'une rupture de la connexion transport ? Comment remédier à la perte d'un enregistrement suite à un problème d'écriture sur disque chez le récepteur ? Tous ces évènements peuvent entraîner des conséquences graves pour les applications, à titre d'exemple : l'abandon du dialogue et sa reprise à zéro. C'est pour donner les moyens aux applications de pallier ce genre de mésaventures que l'on a défini, au dessus du transport, un service utilisable par l'ensemble des applications, le service session (figure II.1).

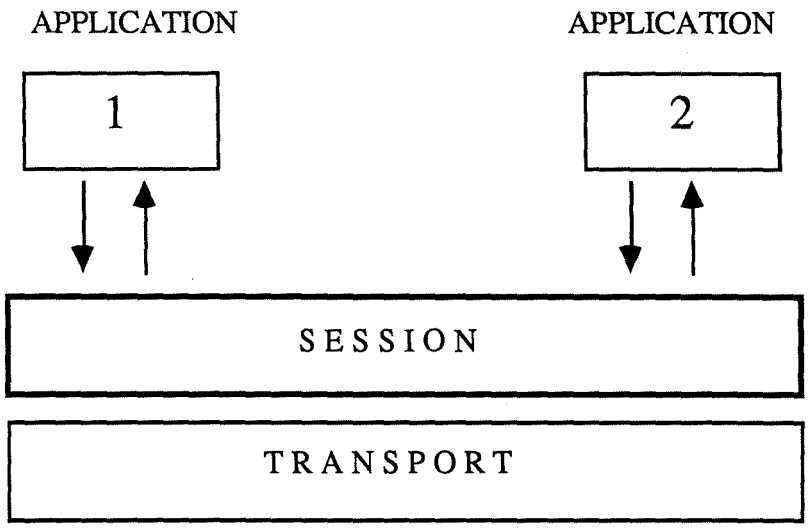


Fig II.1 : situation de la couche session

L'objectif de la couche session est, d'après la norme, "d'offrir les moyens nécessaires à l'échange organisé et synchronisé de données entre utilisateurs du service session coopérants." [ISO8326] Elle permet d'organiser un dialogue, cela signifie : mettre d'accord les utilisateurs sur le déroulement de l'échange, offrir les moyens de le réaliser, vérifier qu'ils ne transgressent pas les conventions de départ. Elle permet aussi de synchroniser l'échange. Autrement dit, à tout moment, les utilisateurs savent se repérer dans le dialogue. De plus, ils ont la possibilité de le reprendre en un point connu de chacun d'entre eux.

Nous tenons à faire remarquer deux caractéristiques de la couche. Premièrement, il ressort assez nettement des objectifs que la session est orientée connexion. La concertation de départ sur les conventions à respecter lors de l'échange, la coopération entre les utilisateurs pour se synchroniser sur le dialogue, induisent intuitivement ce mode de transmission. Deuxièmement, le service est une série d'outils dont l'utilisation est contrôlée par la session. La couche ne réalise pas un service transparent aux utilisateurs. Ce sont ces derniers qui prennent la décision de manipuler les outils.

5 Présentation des concepts

Nous présentons les quatre principaux concepts de la couche session, à savoir :

- la structuration
- le jeton
- l'interruption et la reprise du dialogue
- la négociation.

5.1 Concept de structuration

Nous avons vu, lors de la définition du service, qu'il permet aux utilisateurs de se synchroniser sur l'état d'avancement de l'échange. Pour ce faire, la couche offre des moyens de découpe du dialogue en structures, grâce à la pose de points de repère. Nous allons voir les deux types de structure, à savoir :

- l'unité de dialogue
- l'activité.

5.1.1 L'unité de dialogue

Une unité de dialogue est une partie de l'échange clairement séparée des autres. C'est une décomposition du flux de données. Un point de synchronisation majeur est un point de repère numéroté et inséré dans le flux, permettant de séparer les unités de dialogue. Dans sa représentation la plus simple, une unité de dialogue est une suite de données envoyées d'un utilisateur à un autre, délimitée par deux points de synchronisation majeurs. Un point de synchronisation mineur est un point de repère qui permet aux utilisateurs de s'informer sur l'état d'avancement de l'échange. Il découpe une unité de dialogue en sous-unités (figure II.2).

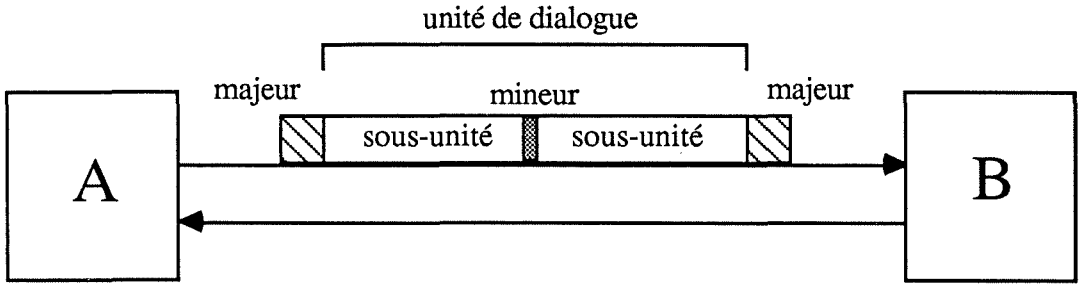


Fig II.2 : représentation d'une unité de dialogue

Imaginons l'envoi d'un texte entre deux utilisateurs, A et B. Un texte est constitué de pages comprenant une ou plusieurs lignes. A désire s'arrêter après chaque page pour permettre à B d'effectuer un traitement. De plus, il attache un caractère particulier à la structuration des pages en lignes : il désire découper ses pages en unités synchronisables. C'est précisément pour réaliser un échange de ce type que l'on se sert de points de synchronisation. Dans notre exemple, une page correspond à une unité de dialogue. A la fin de l'envoi d'une page, l'émetteur pose un point de synchronisation majeur et en attend la confirmation par le récepteur. Ce dernier, lorsqu'il reçoit le point, exécute son traitement. Ensuite, il doit confirmer le point. La réception de la confirmation permettra à l'émetteur d'envoyer la page suivante. Lors de l'envoi d'une page, l'émetteur signale à son récepteur la fin d'une ligne par un point de synchronisation mineur et il continue le transfert jusqu'à la fin de la page. La figure II.3 illustre le transfert des pages dans le temps. Le récepteur peut signaler quelles lignes il a déjà reçues, par la confirmation d'un des points de synchronisation mineurs. Du point de vue de la session, cette confirmation n'est nullement obligatoire.

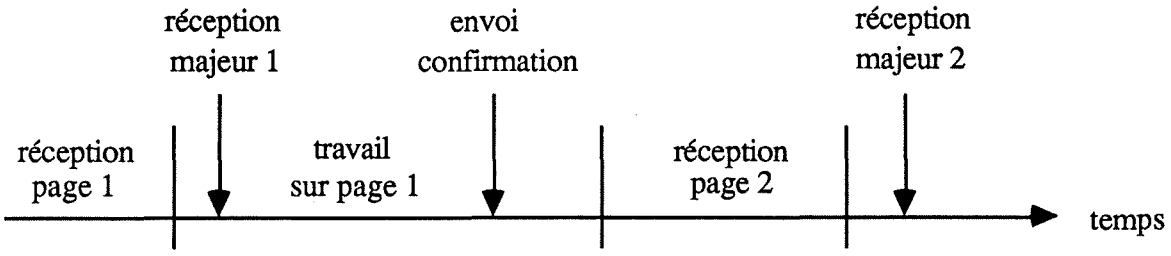


Fig II.3 : transfert des pages dans le temps (récepteur)

Les mécanismes de structuration sont, rappelons-le, de simples outils. Un point de synchronisation majeur sert à marquer une pause dans le dialogue : il bloque l'émetteur du point jusqu' à l'arrivée de la confirmation. De plus, c'est un point de repère numéroté : il permet de vérifier la séquence de la numérotation et de reprendre le dialogue (voir resynchronisation). Un point de synchronisation mineur permet aux utilisateurs de s'informer sur l'avancement du transfert. De plus, tout comme le point de synchronisation majeur, il offre les avantages d'un point de repère numéroté. Enfin, toute valeur sémantique que l'application attache à un point de synchronisation échappe complètement à la couche. C'est ainsi qu'un point de synchronisation mineur peut tout aussi bien délimiter une ligne ou une page dans deux contextes différents. Ceci implique qu'il en existe un grand nombre d'utilisations possibles.

5.1.2 L'activité

L'activité est une unité logique de travail identifiable qui comprend une ou plusieurs unités de dialogue. En d'autres mots, elle regroupe sous une identité, une ou plusieurs parties de l'échange. Ses deux caractéristiques principales sont les suivantes :

- une seule activité est permise à la fois sur une connexion session
- il est possible d'interrompre et de reprendre une activité sur une même connexion ou sur une connexion ultérieure.

Reprenons l'exemple précédent en émettant une nouvelle hypothèse : A désire envoyer deux textes à B. Afin de ne pas mélanger les deux textes, A signale à B le début et la fin d'un texte. Etant donné la longueur d'un texte, il peut être intéressant d'interrompre son envoi et de le reprendre par la suite. Dans ce contexte, l'envoi d'un texte est une activité. En effet, l'émission de toutes les pages d'un même texte correspond à une unité logique de travail. Avant de commencer cette transmission, A ouvre une activité qu'il identifie. Il transmet alors le texte. Il peut s'interrompre, par exemple, pour exécuter une activité plus prioritaire. Par la suite, il a la possibilité de reprendre l'envoi du texte. Une fois l'envoi terminé, il clôture l'activité. Les figures II.4 et II.5 illustrent cet échange.

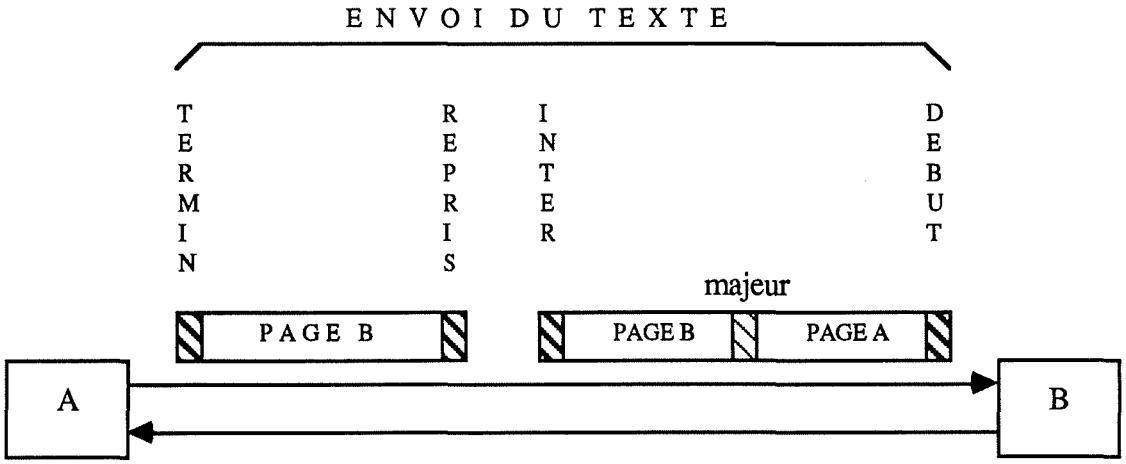


Fig II.4 : transfert d'un texte

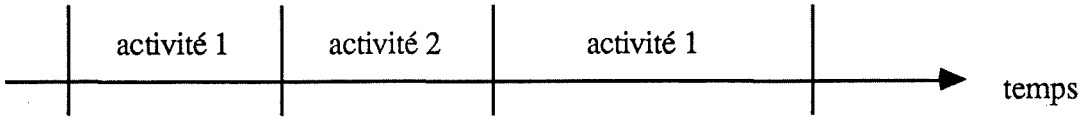


Fig II.5 : transfert du texte dans le temps

Il est important de remarquer qu'un outil de structuration ne possède aucune valeur sémantique intrinsèque. Il peut donc y correspondre un grand nombre d'utilisations. On peut regrouper dans une activité, l'ensemble des manipulations exercées sur les éléments et les attributs d'un même fichier, ou bien, considérer comme dans notre exemple qu'une activité correspond à l'envoi d'un texte. L'objectif de cette structuration est de regrouper des éléments du dialogue et d'identifier ces regroupements, afin de pouvoir les interrompre et les reprendre le cas échéant, sur une même connexion ou sur une connexion ultérieure. Elle contraint les utilisateurs à travailler sur une seule activité à la fois. Dès que les caractéristiques d'une activité sont respectées par une partie du dialogue, on peut se servir de cette structuration et jouïr des mécanismes ainsi que des contrôles qui y sont attachés.

5.2 Concept du jeton

Rappelons que la couche session permet d'organiser un dialogue. Elle offre la possibilité de définir des règles d'utilisation de certains de ses services. Par exemple, les utilisateurs peuvent convenir d'utiliser le service d'envoi de données à l'alternat. C'est par l'intermédiaire d'un "jeton" que la couche session permet de réaliser une telle convention.

Un jeton, comme le définit la norme, "est un attribut de connexion session qui est attribué dynamiquement à un utilisateur à la fois, pour lui permettre de faire un usage exclusif de certains services", à savoir : l'envoi de données, la terminaison de la connexion, les services de structuration.

Reprenons notre exemple en émettant deux nouvelles hypothèses. Premièrement, A et B désirent tous deux envoyer un texte. Deuxièmement, un seul texte peut être transmis à la fois. L'échange devra donc se faire à l'alternat. Au départ, A possède seul le droit d'émettre un texte. Il détient alors "le jeton". Toute tentative de B de transmettre des données sera empêchée par la couche session. B va donc demander à A la permission d'émettre : il lui demande "le jeton". Lorsque A décide d'accorder ce droit, il interrompt sa transmission et il cède la parole à B : il lui donne "le jeton". Les rôles sont maintenant inversés.

La norme définit quatre jetons. Le jeton de données permet de réaliser un dialogue de type half duplex contrôlé par le fournisseur session. Un seul utilisateur peut ainsi envoyer des données. Toute tentative d'envoi par l'autre utilisateur sera empêchée par la couche. La norme attache deux jetons aux services de structuration du dialogue : le jeton de synchronisation mineur, le jeton majeur et d'activité. Ils définissent un rapport maître-esclave entre les utilisateurs désireux de se synchroniser : celui qui détient le jeton structure le dialogue. La norme définit un quatrième jeton, celui de la terminaison négociée. Il permet à son détenteur de demander la terminaison de la connexion. L'autre peut alors refuser.

Dans notre exemple, l'utilisateur qui détenait le droit d'envoyer un texte possédait à la fois trois jetons : le jeton majeur et d'activité lui permettant de commencer l'envoi d'un texte et de signaler les fins de page, le jeton mineur lui donnant la possibilité de signaler les fins de ligne, le jeton de données lui conférant le droit d'envoyer le texte en lui-même. Cependant, on peut imaginer une toute autre situation. Prenons le cas où un utilisateur désire interroger une base de données. Pour cela, il ouvre une activité et envoie une

question. Afin de permettre à son correspondant de lui répondre, il lui cède le jeton de données. Désireux de contrôler la structuration de l'échange, il garde le jeton majeur et d'activité. Deux jetons se trouvent alors dans des mains différentes.

Comme l'illustre ce dernier exemple, l'ensemble des jetons n'est pas obligatoirement détenu par un seul utilisateur. Toutefois, la norme établit des liens entre ces jetons. Elle impose ainsi certaines restrictions à l'utilisation des services qui y sont attachés. Le détail de ces restrictions n'entre pas dans le cadre de ce chapitre.

Remarquons enfin que tous les jetons ne sont pas nécessairement présents lors d'un dialogue (dans notre premier exemple, nous n'utilisons pas le jeton de terminaison négociée). Lorsque les utilisateurs désirent réaliser un dialogue de type half duplex, structurer un échange, négocier la terminaison, les jetons correspondant sont dits "disponibles". Ils sont alors attribués à un seul utilisateur à la fois (séparément ou non). Lorsque les jetons de données et de terminaison négociée sont indisponibles, les services correspondant peuvent être employés par les deux utilisateurs à la fois. Par contre, si un jeton lié à la structuration est indisponible, le service correspondant ne peut être utilisé par aucun des deux interlocuteurs.

5.3 Concept d'interruption et de reprise du dialogue

Lors de la définition de la norme, nous avons introduit la notion de reprise du dialogue en un point préalablement défini. Nous allons aborder ici les différents concepts liés à l'interruption et à la reprise du dialogue, aux deux niveaux de structuration vus précédemment.

5.3.1 Niveau unité de dialogue

Reprenons notre exemple. Lors d'un transfert d'un texte, le récepteur perd une ligne suite à un problème d'écriture sur disque. Il est évidemment plus intéressant de reprendre la transmission avant cette ligne plutôt qu'à partir du début du texte. C'est entre autres pour résoudre ce genre de problème qu'a été créé le concept de resynchronisation.

La resynchronisation c'est la reprise du dialogue en un point de synchronisation, avec une nouvelle affectation des jetons. Elle permet de reprendre le transfert dans l'unité de dialogue courante (option redémarrage), à un nouveau point de synchronisation (option abandon), ou enfin, en un ancien point de synchronisation quelconque (option choix de l'utilisateur).

Développons l'exemple du transfert de textes. A envoie un texte à B. Admettons que lors du traitement de la ligne 2, le récepteur B découvre une erreur (cfr plus haut). Il veut demander à A de reprendre le dialogue juste avant cette ligne. Pour ce faire, B émet une demande de resynchronisation sur le point de synchronisation mineur 2, avec option redémarrage. L'émetteur A, après confirmation de la resynchronisation, reprendra le transfert au point indiqué. La figure II.6 illustre ce dialogue. Prenons maintenant l'hypothèse que l'émetteur détecte une erreur : un problème survenant lors de la lecture de la ligne 5 lui fait sous-entendre que les deux dernières lignes envoyées (4 et 3) peuvent contenir des erreurs. Il envoie alors une demande de resynchronisation option redémarrage portant sur le point de synchronisation mineur 3. Après réception de la confirmation, il reprendra le transfert en ce point (figure II.7).

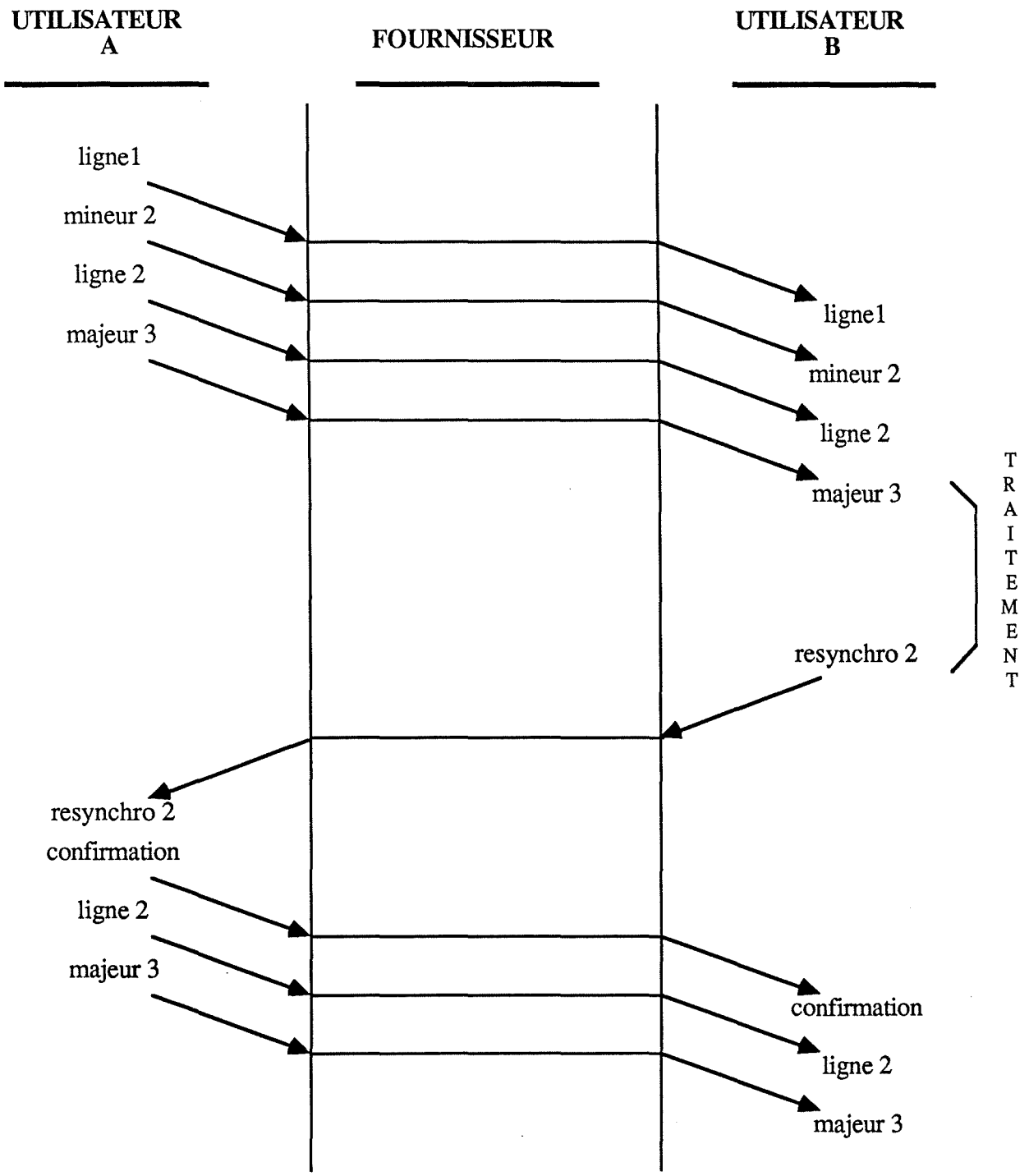


Fig II.6 : resynchronisation par le récepteur

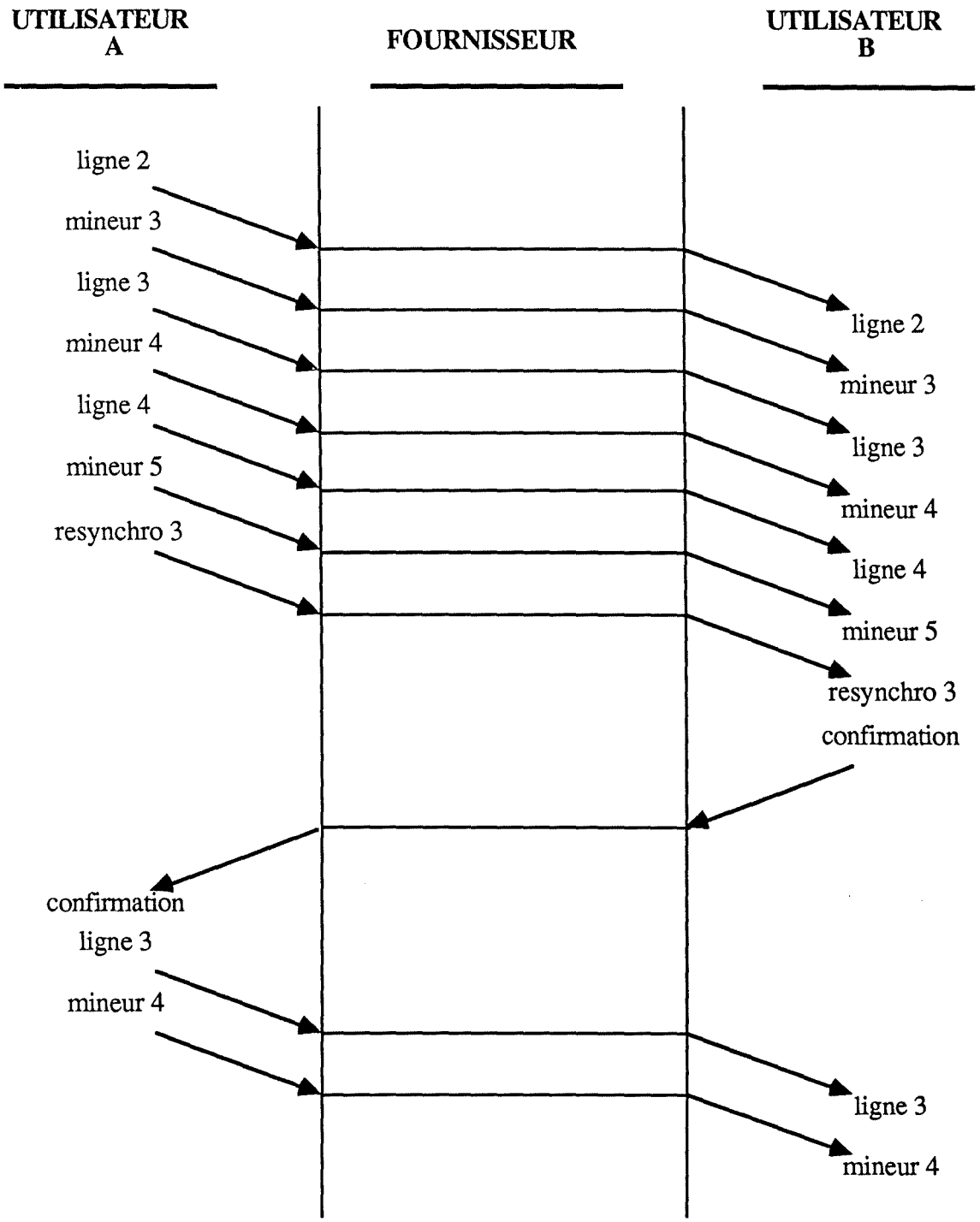


Fig II.7 : resynchronisation par l'émetteur

La resynchronisation permet donc la reprise du dialogue en un point de synchronisation connu des deux utilisateurs. Elle peut marquer une pause chez l'émetteur.

La norme définit trois options : redémarrage, choix de l'utilisateur, abandon. Les deux premières diffèrent entre elles au niveau du contrôle exercé par la couche session. Si les utilisateurs désirent que celle-ci vérifie l'appartenance du numéro de resynchronisation

à la dernière unité de dialogue, ils emploient l'option redémarrage. Dans ce cas, toute tentative de resynchronisation dans une unité de dialogue antérieure est interdite par la couche session. Sinon, ils emploient l'option choix de l'utilisateur. Le fournisseur vérifie alors que le numéro correspond à un point de synchronisation déjà posé. L'option abandon permet de reprendre le dialogue en un nouveau point fourni par la session. La portée de l'abandon (sur la dernière unité ou sur l'ensemble des unités de dialogue) doit être convenue entre utilisateurs.

N'oublions pas que la resynchronisation permet aussi une nouvelle affectation des jetons disponibles. Prenons l'exemple d'un utilisateur se voyant refuser un service par la couche session parce qu'il ne possède pas le jeton adéquat. Si son interlocuteur s'obstine à ne pas répondre à ses demandes de jetons, il peut interrompre le dialogue par une demande de resynchronisation, en signifiant à son correspondant le jeton qu'il désirerait obtenir.

5.3.2 Niveau activité

Nous avons vu précédemment qu'il est possible d'interrompre et de reprendre une activité : c'est une des caractéristiques principales de ce concept. L'objectif est d'en permettre le morcellement et la prolongation sur plusieurs connexions. De plus, il est possible de l'abandonner. Le but est alors, pour un utilisateur, de signaler à son correspondant que l'ensemble du dialogue contenu dans l'activité courante doit être ignoré.

Reprenons l'exemple du transfert de textes en y ajoutant deux hypothèses. Premièrement, une erreur grave peut se produire lors du transfert, impliquant l'abandon de la totalité du texte courant (type I). Admettons qu'elle provienne du mauvais fonctionnement d'un utilisateur ou du fournisseur du service session, suite à une erreur de programmation. Lorsqu'un événement de type I est détecté par l'utilisateur maître A, il emploie le service d'abandon d'activité. Ceci clôture anormalement et définitivement le transfert en cours. Deuxièmement, supposons qu'un certain crédit de temps est alloué pour chaque connexion. Une fois ce crédit dépassé, la transmission doit être interrompue (type II). Lorsque A détecte un événement de type II, il doit interrompre l'activité courante. Lorsque la confirmation de B arrivera, A pourra couper la connexion. Ensuite, lorsqu'il lui sera à nouveau permis d'ouvrir un dialogue, A initialisera une connexion et reprendra son activité en un point de synchronisation convenu avec B.

L'utilité des mécanismes décrits plus haut est de permettre aux utilisateurs de réagir, au niveau activité, face à un événement. Ainsi que notre exemple l'illustre, plusieurs catégories d'événements peuvent occasionner une terminaison anormale d'activité : ceux qui impliquent l'abandon du dialogue, et ceux qui nécessitent l'interruption de l'activité courante. Ceci dépend des conventions prises par les deux utilisateurs.

5.4 Concept de négociation

Lorsqu'un utilisateur veut dialoguer avec un autre, il doit ouvrir une connexion de niveau session. Il propose pour cela deux types de desiderata : les uns en matière de connexion, les autres en matière de session. Les premiers concernent les caractéristiques d'une connexion : le débit, le temps de transit dans la couche ... Les deuxièmes concernent le type de dialogue qu'il souhaite instaurer, par exemple : un transfert full duplex constitué de plusieurs unités logiques de travail structurées, avec possibilité de resynchronisation. L'utilisateur va pour cela proposer une liste de services qu'il désire utiliser. Notons que certains services sont liés à des jetons. Une affectation initiale des jetons doit être établie. De plus, au service de synchronisation est attachée une série de numéros. Un numéro intitial devra être décidé.

Lorsqu'un utilisateur est appelé, il exprime son accord quant aux desiderata de l'appelant, en proposant les mêmes caractéristiques de connexion et de dialogue. Il signale son désaccord par une proposition différente. En cas de désaccord sur les termes de l'échange, un compromis doit être conclu entre les utilisateurs.

Une troisième partie intervient dans cette discussion : le fournisseur. Il dispose lui aussi de certaines caractéristiques de connexion et de session. Rappelons que la couche repose sur le transport. A un moment donné, les possibilités de connexion transport peuvent être réduites pour cause d'engorgement. Ensuite, il se peut qu'un fournisseur session ne dispose pas, par exemple, des mécanismes de gestion d'activité. C'est un choix d'implémentation qui peut être influencé par des contraintes de place mémoire ou de performance de communication. Les caractéristiques du fournisseur doivent être compatibles avec les desiderata des utilisateurs (figure II.8).

L'objectif de la négociation est de résoudre l'ensemble de ces problèmes.

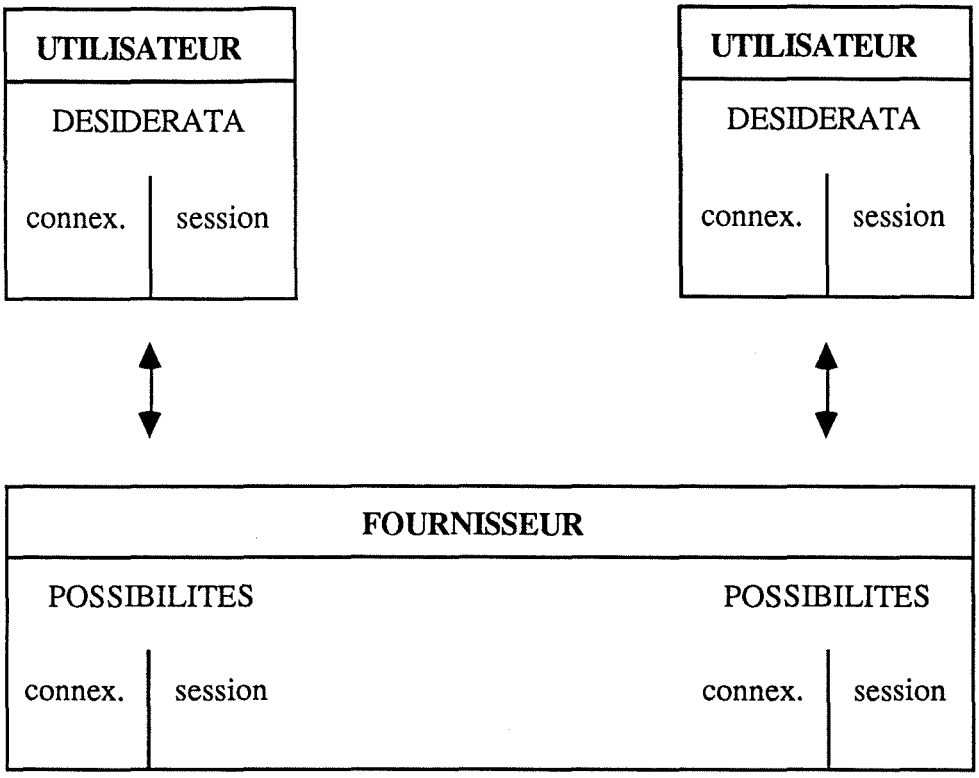


Fig II.8 : confrontation tripartite

La négociation, c'est la réalisation d'un compromis entre les propositions des deux utilisateurs en matière de connexion session, et les possibilités offertes par le fournisseur. Les possibilités du fournisseur dépendent de l'éventail de services de niveau session qu'il peut mettre à la disposition de ses utilisateurs et des disponibilités de la couche transport à l'ouverture de l'échange. Une proposition d'utilisateur comprend les caractéristiques de connexion et la liste des services qu'il désire employer. A certains services sont attachés des jetons qui devront être attribués. De plus, si la synchronisation est désirée, le premier numéro de série pourra être décidé.

Reprenons notre exemple de transfert de textes. A ouvre une connexion avec B afin de lui transférer un texte. Il n'a aucun desiderata particulier quant aux caractéristiques de connexion. Il propose comme service : la synchronisation majeure et mineure, la resynchronisation. Il dispose d'une couche session offrant l'ensemble de ces services. Il désire posséder l'ensemble des jetons. De plus, il souhaiterait que la série de numéros de points de synchronisation commence à 1. B possède une couche session comprenant l'ensemble des services de ce niveau. Il est capable d'utiliser tous ses services. Il a pour habitude de démarrer la série de numéro à 0. Nous verrons, à travers les points suivants, comment un compromis peut être réalisé.

La norme, afin de résoudre plus facilement le problème de la négociation, introduit trois notions : l'unité fonctionnelle, le sous-ensemble, la qualité de service (QOS). Nous les aborderons dans cet ordre.

5.4.1 Les unités fonctionnelles

La norme offre un ensemble de services. La possibilité est donnée aux utilisateurs de sélectionner uniquement les services qui sont nécessaires à leur application, et de négocier ce choix. Afin de faciliter la négociation ainsi que toute référence à d'autres normes, les concepteurs ont défini la notion d'unité fonctionnelle.

Une unité fonctionnelle est un regroupement logique de services. La norme définit les unités fonctionnelles suivantes : le noyau, la transmission de données expresses, de données typées, d'information de capacité, la synchronisation mineure, la synchronisation majeure, la gestion d'activité, la resynchronisation, la transmission half duplex, la transmission full duplex, la terminaison négociée et la signalisation d'anomalie. Le noyau comprend les services minimaux session : l'ouverture, la fermeture et la coupure de la connexion, l'envoi de données. La transmission de données expresses correspond à la notion habituelle de flux express que l'on retrouve dans d'autres couches. Rappelons ses caractéristiques : une donnée envoyée sur le flux express peut arriver avant une donnée envoyée en même temps sur le flux normal, mais pas après. La transmission de données typées correspond à un transfert qui se veut différent de l'envoi de données normales. L'échange d'informations de capacité permet un envoi confirmé de données particulières. La signalisation d'anomalie permet de signifier aux utilisateurs la survenance d'une anomalie chez le fournisseur. Elle permet aussi de s'informer entre utilisateurs de l'apparition d'une anomalie. Les services correspondant aux autres unités fonctionnelles ont été introduits préalablement.

Reprenons l'exemple présenté ci-dessus. A sélectionne les services de synchronisation mineure et majeure, et celui de resynchronisation. Il propose, lors de sa demande de connexion, ces trois unités fonctionnelles. Aux unités fonctionnelles de synchronisation sont attachés des jetons. Pour chacune d'entre elles, A peut attribuer le jeton à lui-même ou à son correspondant. Il peut aussi laisser cette initiative à B. Dans notre cas, il s'attribue les deux jetons. Il signifie aussi à B la valeur du numéro initial de synchronisation (dans notre exemple, sa valeur est égale à 1). Quant à B, il est prêt à proposer l'ensemble des unités fonctionnelles. Il laisse à A le choix du numéro de synchronisation. S'il désirait au contraire imposer son numéro, ce dernier l'aurait emporté

sur A ; c'est un choix de la norme. Le résultat de la négociation des propositions de services est développée au point suivant.

L'unité fonctionnelle est donc le moyen dont disposent les utilisateurs de la couche pour exprimer leurs besoins en service. Une unité fonctionnelle permet de référencer de façon générique un service. Prenons l'exemple de la synchronisation majeure. Ce service comprend toute une série de primitives que doivent utiliser les applications afin de réaliser leur synchronisation, à titre d'exemple : la pose et la réponse à la pose d'un point de synchronisation majeur; la demande et la cession de jeton. Par la simple demande de cette unité fonctionnelle, sauf si la négociation en décide autrement, les utilisateurs auront à leur disposition l'ensemble de ces primitives. Ces regroupements permettent ainsi aux normes décrivant des couches qui utilisent les services de niveau session, de référencer facilement ces services. Nous en verrons quelques exemples par la suite.

5.4.2 Les sous-ensembles

L'utilisateur, lors de l'ouverture de la connexion, propose donc un ensemble d'unités fonctionnelles. Des restrictions en empêchent certaines combinaisons. Par exemple, on ne peut réaliser à la fois un dialogue half et full duplex. La norme impose aussi que le noyau soit toujours proposé. Sans cette unité fonctionnelle toute communication est impossible. L'utilisateur doit en tenir compte. La session définit pour cela des combinaisons valides : des "sous-ensembles". Afin de faciliter le choix des utilisateurs et de standardiser l'emploi de la couche, les concepteurs ont proposé des "sous-ensembles types".

La norme définit un sous-ensemble comme étant une combinaison d'unités fonctionnelles respectant certaines conditions. De plus, elle propose trois sous-ensembles particuliers : le BCS (Basic Combined Subset), les BSS (Basic Synchronized Subset), le BAS (Basic Activity Subset). Le BCS comprend le noyau, la transmission half et full duplex (au choix). Le BSS comprend le noyau, la transmission half duplex, la terminaison négociée, la transmission de données typées, la synchronisation mineure et majeure, la resynchronisation. Le BAS comprend le noyau, la transmission half duplex, de données typées, d'information de capacité, la signalisation d'anomalie, la synchronisation mineure, la gestion d'activité.

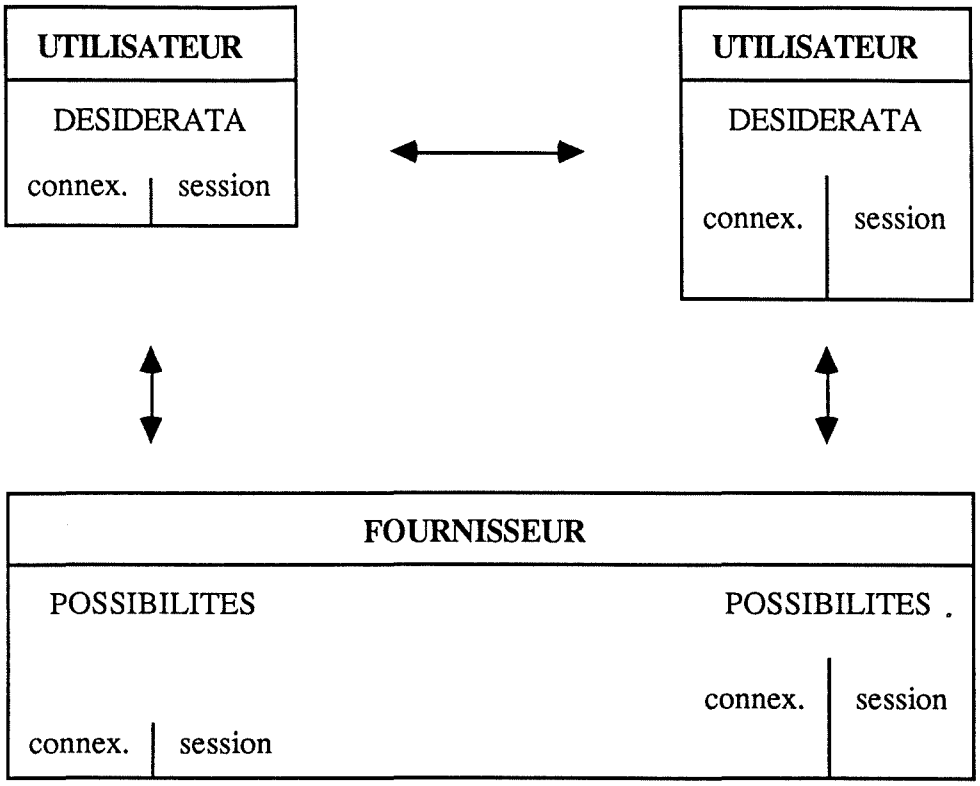


Fig II.9 : illustration de la négociation

Reprenons l'exemple décrit plus haut. L'utilisateur A propose donc un sous-ensemble composé des unités fonctionnelles noyau, half duplex, de synchronisation mineure et majeure, de resynchronisation. On peut remarquer qu'il correspond à peu près à BSS. Rappelons que l'entité session utilisée par A ne comprend que ces services. B était prêt à tout dialogue . Il propose donc un sous-ensemble composé de l'ensemble des unités fonctionnelles, hormis la transmission full duplex (pour des raisons de validité). Rappelons que l'entité session utilisée par B comprend l'ensemble des services. La figure II.9 illustre cette situation. Le fournisseur est composé des deux entités. Il ne peut réaliser que l'intersection des services supportés par ces entités. Lors de la phase de connexion, il vérifie la validité des combinaisons proposées par A et B. De plus, il s'assure qu'il peut réaliser localement les services demandés. Les sous-ensembles proposés par A et B sont acceptés par le fournisseur. La négociation résultera en l'intersection des sous-ensembles proposés par les deux utilisateurs. Dans notre exemple, il correspond à la proposition de A.

Un sous-ensemble particulier reflète donc l'ensemble des desiderata d'un utilisateur en matière de service. Ce regroupement d'unités fonctionnelles est contrôlé par la couche (cohérence de la combinaison, disponibilité des services demandés). La négociation résulte en un sous-ensemble qui est l'intersection des deux combinaisons valides proposées. C'est

un compromis entre les services demandés par les utilisateurs et leur fournisseur. Remarquons qu'un concepteur de logiciel session peut n'implémenter qu'un sous-ensemble particulier de services.

Par ailleurs, la norme propose trois sous-ensembles types. Le BCS est destiné à des applications qui ne nécessitent pas de structuration. Il propose un service minimal session. Le BSS est destiné à des applications nécessitant les services de synchronisation. Il convient, comme le souligne Wendy Rauch-Hindin [RAU 86], au transfert de fichiers ou de grandes quantités de données. Enfin, le BAS, d'après Willard F. Emmons et A. S. Chandler [EMM 83], était destiné, au départ, aux applications télématiques du CCITT, tel que le télétex. Rappelons que la norme T62 a été intégrée au protocole session de l'ISO. Ce sous-ensemble convient aussi à d'autres applications qui nécessitent une structuration en activités. Pour terminer, notons que ce sont des sous-ensembles de base : une application désirant une structuration maximale de son dialogue peut proposer les unités fonctionnelles du BAS complétées de la synchronisation majeure.

5.4.3 La qualité du service session

Nous avons vu que, lors de l'ouverture de la connexion, l'utilisateur a le droit de préciser certaines caractéristiques de connexion. Celles-ci sont regroupées sous l' appelation qualité du service session. Nous citons à titre d'exemple :

- la protection de la connexion
- les priorités des connexions
- le taux d'erreur résiduel pendant la phase de transfert
- le débit dans chacun des sens de transfert
- le temps de transit d'une demande de service dans le fournisseur
- le transfert avec optimisation du dialogue
- le contrôle étendu.

Seules les deux dernières caractéristiques ont retenu notre attention : elles concernent chacune un mécanisme particulier du fournisseur session. Le transfert avec optimisation du dialogue concerne l'optimisation de l'utilisation du service d'envoi de données de niveau transport. Elle est réalisée par la concaténation d'éléments de protocole session. Le contrôle étendu permet l'utilisation du flux express transport pour certains éléments du protocole session. Cela sert entre autres à débloquer au plus tôt un utilisateur suite à la pose d'un point de synchronisation majeur. Nous reviendrons sur ces deux mécanismes dans le chapitre suivant.

La négociation de ces deux caractéristiques est réalisée très simplement. Si une partie ne propose pas un mécanisme, il n'est pas adopté. Dans notre exemple, aucun des deux utilisateurs n'exprime de souhait particulier en matière de connexion. Aucun de ces mécanismes ne sera donc réalisé par le fournisseur.

6 Justification de la couche session ISO

6.1 Utilité et utilisation

Résumons l'utilité principale de la couche session. Cette couche est un "programme à la carte" : les utilisateurs peuvent sélectionner les services qu'ils désirent employer. Elle permet d'identifier une connexion. Les utilisateurs ont ainsi la possibilité de différencier un dialogue particulier d'un autre. Rappelons les principaux services offerts par ce niveau : l'établissement d'un compromis entre les desiderata des deux utilisateurs et les possibilités d'un fournisseur particulier, la structuration du dialogue, la pose de points de repère sur le flux de données, la synchronisation de l'échange par la confirmation de ces points, la possibilité de reprise du transfert en un point convenu à l'avance. Rappelons aussi que le fournisseur session contrôle l'utilisation de ses services. Cela revient à dire qu'il vérifie, premièrement, le droit de demander un service en fonction de sa disponibilité : celle-ci résulte de la négociation et de l'existence d'un jeton. Deuxièmement, il vérifie les paramètres qui accompagnent une demande de service, par exemple : lors d'une demande de resynchronisation option redémarrage, le numéro de série proposé doit appartenir à un intervalle de valeurs particulier. Finalement, il vérifie l'enchaînement de toutes les demandes en fonction de règles définies. A titre d'exemple, rappelons qu'une seule activité à la fois peut être ouverte sur une connexion.

Citons maintenant quelques utilisations particulières de ce niveau définies dans d'autres normes. Rappelons d'abord l'analogie qui existe entre le sous-ensemble BAS et la norme T62. Lors de l'intégration de cette norme, les notions d'envoi de document télétex et d'activité ont été mises en correspondance. Le protocole de messagerie électronique X400 du CCITT utilise lui aussi le BAS. Le RTS (Reliable Transfer Server) de la norme X410 référencée par la norme X400 utilise en fait l'ensemble des unités fonctionnelles proposées par BAS, hormis l'unité fonctionnelle de données typées et l'unité fonctionnelle d'échange d'informations de capacité. Le RTS n'est pas l'unique CASE (Common Application Service Element) à faire référence au sous-ensemble BAS. Le CCR (Commitment Concurrency and Recovery), dans sa version 'working draft' utilise les primitives de gestion d'activité et de données typées. Hormis la non-utilisation de la synchronisation mineure, cela se rapproche sensiblement du sous-ensemble BAS. L'application FTAM (File Transfer Access and Management) ne fait pas directement référence à un sous-ensemble précis. Les services session nécessaires au protocole FTAM sont les services de synchronisation et de resynchronisation. Cela semblerait indiquer une

tendance vers le BSS. Toutefois, l'application FTAM utilise le CASE CCR. Dans l'état actuel de notre investigation, nous ne pouvons dès lors rien affirmer quant à l'utilisation par FTAM d'un quelconque sous-ensemble. Nous essayerons, par la suite, de spécifier les besoins en session de diverses "caricatures" d'applications standards telles que FTAM, JTM (Job Transfer and Manipulation), VT (Virtual Terminal). Cela pourra a priori, selon certaines hypothèses, nous indiquer vers quel type de sous-ensemble ces applications peuvent se diriger.

6.2 Justification

Nous allons maintenant tenter de justifier ce que nous considérons comme les principales caractéristiques de la couche , à savoir :

- programme à la carte et sous-ensembles
- mécanisme de structuration et d'organisation.

6.2.1 Programme à la carte et sous-ensembles

La session offre un ensemble de services que les utilisateurs sélectionneront en fonction des besoins précis de leurs applications. Cette sélection peut engendrer un service très réduit, à l'exemple de BCS, ou alors, particulièrement puissant dans le cas où, par exemple, l'ensemble des unités fonctionnelles hormis l'unité fonctionnelle full duplex ont été choisies.

De par cette approche, la session ISO se veut extensible, suffisamment générale et spécialisable. Elle se veut extensible afin de ne pas interdire toute extension future de la gamme des utilisateurs du service qu'elle offre. Elle se veut suffisamment générale, pour pouvoir convenir à toute cette gamme d'utilisateurs. Elle se veut spécialisable afin de convenir précisément à une utilisation particulière.

Un exemple concret de cette extensibilité est l'addendum sur la synchronisation symétrique. Il permet une structuration des deux flux de données existant lors d'une communication full duplex. Il offre ainsi la possibilité de se resynchroniser sur ces deux flux. Les concepteurs de la norme session s'étaient en fait aperçus que la synchronisation proposée jusque là était purement de nature half duplex. La proposition des sous-ensembles BCS, BSS et BAS, et non leur imposition, renforce l'idée de généralité.

Rappelons qu'un fournisseur session particulier peut n'offrir qu'une partie de ces services. De plus, ainsi que l'indique la première lettre des abréviations BCS, BSS, et BAS, ce sont des sous-ensembles de base. Cela signifie qu'une application d'où émerge la notion d'unité logique de travail, peut tout aussi bien nécessiter une structuration plus forte de ses activités que celle prévue dans BAS. Rien n'empêche aux concepteurs de ce genre d'application, de demander un 'sous-ensemble activité étendu' comprenant, en plus du sous-ensemble BAS, l'unité fonctionnelle de synchronisation majeure et, de façon optionnelle, celle de resynchronisation.

6.2.2 Mécanisme de structuration et d'organisation

La couche session n'offre que des mécanismes de synchronisation et d'organisation du dialogue. C'est à l'utilisateur qu'appartient l'initiative de poser des points de synchronisation, de les confirmer, de se resynchroniser. La couche session quant à elle vérifie l'utilisation correcte des primitives correspondantes (à savoir : le droit d'émettre, les valeurs des paramètres et l'enchaînement des primitives) et informe l'utilisateur en cas de non respect des conventions.

On aurait pu imaginer une couche session qui non seulement offrirait ces mécanismes, mais en plus, les mettrait en oeuvre elle-même. Il suffirait pour cela de signifier au fournisseur session la fréquence à laquelle les points de synchronisation doivent être envoyés. Le fournisseur du service session prendrait, lors de la phase de transfert, l'initiative de poser ces points et d'y répondre. En cas de problème, il resynchroniserait, soit de sa propre initiative, soit selon l'avis des utilisateurs. C'est dans cet ordre des choses que s'inscrit la démarche de Popescu . [POP 84]

Le point de vue ISO est celui d'une synchronisation session sur-mesure, conformément aux desiderata des utilisateurs. Le deuxième point de vue est celui d'une synchronisation session prêt-à-porter, prêt-à-tourner, ne tenant pas compte de la spécificité des utilisateurs de la session, et de ce fait, ne s'inscrivant pas conceptuellement dans le niveau 5 de l'architecture ISO.

La réalisation du deuxième point de vue allège énormément la procédure de service. De plus, il apporte une indépendance plus forte entre la couche session et les couches supérieures. En effet, on supprimerait alors les primitives de synchronisation et éventuellement de resynchronisation, ainsi que les conventions qui s'y rapportent. En cours de transfert, l'utilisateur ne pourrait plus intervenir sur ces mécanismes qui se

dérouleraient à l'intérieur de la couche session de façon transparente. Ceci conviendrait à priori particulièrement à un simple transfert massif de données. Cette synchronisation est très comparable à un simple contrôle de transfert de données. Le but en serait d'assurer que les données ont bien été reçues par le correspondant. Nous ne voyons pas comment justifier la présence d'un tel mécanisme au niveau 5 du modèle de référence. Nous le situerions plutôt au niveau transport.

Le point de vue de l'ISO nous semble beaucoup plus défendable. Les primitives session sont utilisées au niveau présentation. Ces primitives ne font en fait que passer à travers la couche présentation. L'interaction n'existe réellement qu'entre le niveau 5 et le niveau 7. L'intervention de l'utilisateur, pour synchroniser et resynchroniser le dialogue, demander ou passer le contrôle, et les avertissements du fournisseur session en cas d'erreur, de problème ou de non respect des conventions d'utilisation de ces services, alourdissent considérablement les services session et présentation. Néanmoins, la valeur sémantique attachée à la structuration n'est connue que de l'application. En effet, si l'on attache une importance sémantique aux points de repère (correspondant par exemple, dans le cadre d'un transfert massif de données, à une découpe en enregistrements et sous-enregistrements), seule l'application connaît ce concept, cette structure, et peut, dès lors, poser au bon moment des points de repère et décider de reprendre à un enregistrement ou sous-enregistrement. La session, elle, n'offre en fait qu'un instrument syntaxique (une suite bien précise de bits intercallés dans le flux de données) et elle en contrôle l'utilisation. Ce point de vue permet, tout en normalisant ces mécanismes, de garder une généralité vis-à-vis des spécificités existant dans la gamme des utilisateurs du service.

7 Conclusion

L'objectif de ce chapitre était de situer clairement la session au sein de la normalisation.

Nous avons d'abord observé l'état de la normalisation. De cette approche, il est apparu que la partie orientée connexion est stabilisée et concrétisée en standards internationaux. Les addenda concernant ce mode de transmission sont en voie de standardisation. La définition d'une norme session orientée non-connexion est toujours à l'étude.

De sa définition, il est apparu que la session ISO n'était qu'un outil de synchronisation et d'organisation du dialogue. Cette définition nous a permis d'introduire les concepts sous-jacents, leur signification et leur portée réelle. Rappelons les quatre principaux concepts :

- la structuration
- le jeton
- l'interruption et la reprise du dialogue
- la négociation.

Nous avons alors tenté de justifier cette couche. Nous en avons pour cela souligné l'utilité principale. Ensuite, les tendances d'utilisation de ses sous-ensembles par d'autres normes ont été présentées. Enfin, nous avons essayé de justifier ses deux principales caractéristiques, à savoir :

- programme à la carte et proposition de sous-ensembles
- mécanisme de structuration et d'organisation.

Nous avons alors pris position en faveur de l'ISO en insistant, premièrement, sur le caractère général, spécialisable et extensible de son point de vue, et deuxièmement, sur la situation, dans l'architecture, des concepts présentés.

III CHAPITRE 2 : REALISATION DE LA COUCHE SESSION

1 Introduction

Dans ce chapitre, nous allons étudier la réalisation de la couche session : la mise en oeuvre particulière au concepteurs de la norme ISO, des concepts que nous avons introduits.

L'objectif de ce chapitre est de décrire les services offerts par la couche session, de les analyser, d'en dégager des critères d'utilisation, et d'établir une démarche facilitant le choix d'unités fonctionnelles pour une application particulière.

Pour ce faire, nous analyserons, en un premier temps, les différentes unités fonctionnelles et leurs primitives de service. Nous décrirons aussi le comportement du fournisseur. Ceci permettra d'éclairer les concepts vus au chapitre précédent et d'en souligner les caractéristiques qui ne peuvent ressortir de leur définition.

Dans une deuxième phase, nous étudierons les mécanismes de traitement d'erreurs proposés par la couche. Ce point est fondamental et intrinsèquement lié à l'utilité de ce niveau. En effet, le premier objectif avoué de la session est de proposer des mécanismes d'organisation et de structuration du dialogue qui permettent aux utilisateurs de se resynchroniser le cas échéant (reprise suite à un événement). Le second objectif est de contrôler l'emploi de ces mécanismes, et donc de détecter les erreurs d'utilisation.

Ayant à présent précisé les différents services du niveau session, nous analyserons, en un troisième point, les trois sous-ensembles de base proposés par la norme.

Fort de ces différentes analyses, nous essayerons ensuite de dégager des critères et d'établir une méthode de choix d'unités fonctionnelles, et de les appliquer à des applications standard tels qu'un transfert de message, un transfert de job ...

Après avoir abordé le problème sous divers angles, nous apporterons une évaluation personnelle sur la réalisation de la couche session ISO.

2 Analyse des services

2.1 Introduction

Nous allons analyser les services unité fonctionnelle par unité fonctionnelle. Nous examinerons les restrictions imposées aux combinaisons d'unités fonctionnelles ainsi que les procédures d'utilisation des primitives de service. Nous décrirons aussi le comportement du fournisseur. Autant que possible, nous donnerons les utilisations des services faites par d'autres normes.

Avant d'analyser en détail les services, nous allons préciser quelques notions de base les concernant. Nous reviendrons ensuite sur les conventions générales que les utilisateurs doivent respecter. Après cela, nous introduirons le lecteur au comportement général du fournisseur.

2.1.1 Notions de base

Ainsi qu'il est de coutume dans la description des différents niveaux de l'architecture ISO, les services offerts par une couche sont exprimés en primitives de requête et d'indication, de réponse et de confirmation (figure III.1). Les primitives contiennent des paramètres.

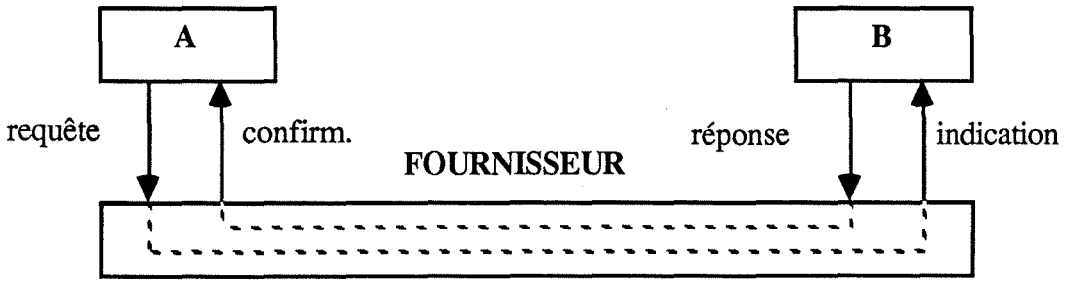


Fig III.1 : primitives de service

Il existe trois types de services : les services confirmés, les services non-confirmés, et les services à l'initiative du fournisseur. Un service est dit confirmé lorsque l'utilisateur l'ayant demandé attend une réponse de son correspondant. C'est un service en quatre temps (il y correspond les quatre primitives citées plus haut). Un service est dit non-confirmé lorsque l'utilisateur qui l'a demandé n'en attend pas de réponse. C'est un service en deux temps (requête, indication). La troisième catégorie de services renferme les

services qui, au contraire des deux catégories précédentes, sont à l'initiative du fournisseur. Ce sont des services en un temps (le fournisseur remet à chacun de ses utilisateurs une primitive d'indication). La figure III.2 illustre ces trois types de service.

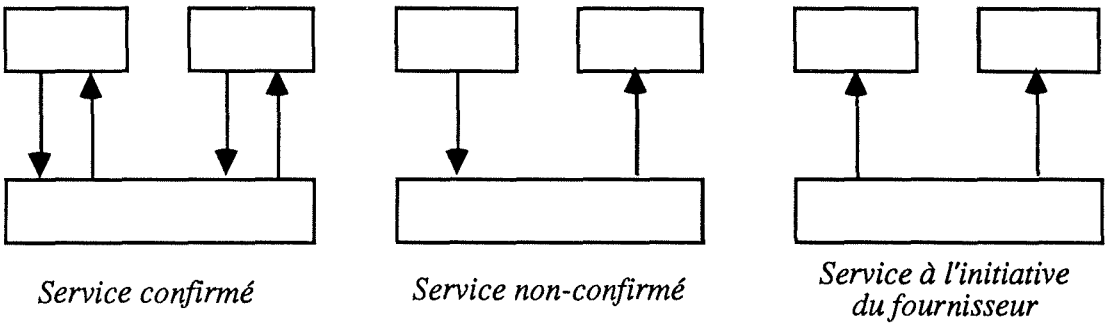


Fig III.2 : types de services

L'échange d'information entre deux couches (N) et (N+1), résultant de l'émission d'une primitive, est formalisé par un (N)-SDU (Service Data Unit). Sa réception par le niveau (N) peut entraîner l'échange d'un ou plusieurs (N)-PDU (Protocol Data Unit) entre deux (N)-entités coopérantes qui réalisent le service de ce niveau (figure III.3).

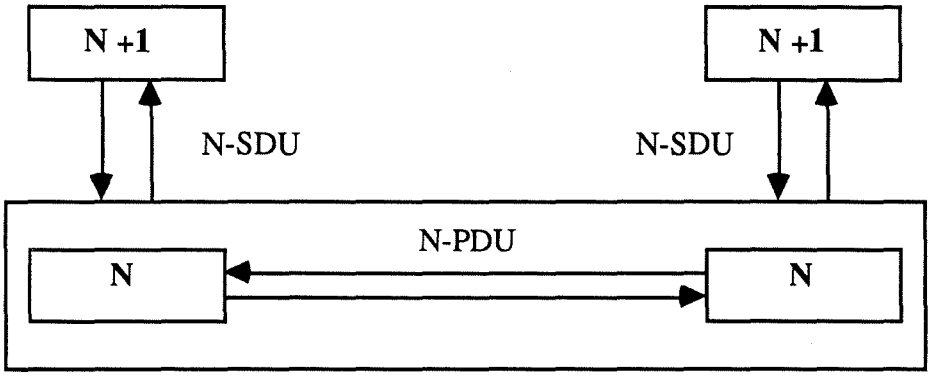


Fig III.3 : échange d'information

Dans le formalisme ISO, une (N+1)-entité accède au service de niveau (N) par un (N)-SAP (Service Access Point). Un (N)-SAP est identifié par une (N)-adresse. Cette (N)-adresse permet aussi à la (N)-entité de déduire la (N-1)-adresse. Lorsque la connexion est réalisée par le fournisseur de niveau (N), ce dernier donne à chaque (N+1)-entité un identificateur local de connexion (N)-CEP (Connection End Point). Les (N+1)-entités peuvent dès lors communiquer grâce à cette connexion.

Nous avons dit que le fournisseur session est réalisé par la coopération de deux entités de niveau 5. Remarquons que, dans la norme, on préfère employer les initiales

SPM (Session Protocol Machine). Une SPM est une "machine abstraite qui effectue les procédures spécifiées dans le texte du protocole". Une entité session peut être décomposée en une ou plusieurs SPM. C'est un choix qui est laissé aux personnes implémentant la couche. Pour des raisons de simplicité, nous considérons qu'à une entité correspond une SPM.

Remarquons que, dans les normes ISO, on retrouve souvent le qualificatif "transparent". Ce qualificatif est une traduction malencontreuse de son homonyme anglais. Sa signification réelle est "opaque". En effet, lorsque l'on dit qu'un service est transparent à un utilisateur, cela signifie que l'utilisateur ne voit pas comment le service est réalisé à l'intérieur du fournisseur, qu'il ne peut agir sur la façon dont le service est réalisé. Une donnée de l'utilisateur est transparente à la couche session lorsqu'elle est "laissée intacte hors de son transfert entre SPM et non-utilisable par celles-ci". [ISO8327]

2.1.2 Conventions générales

Rappelons que les unités fonctionnelles doivent avoir été sélectionnées et retenues lors de la négociation pour que les utilisateurs puissent jouïr des services qu'elles regroupent. Souvenons-nous que certaines restrictions sont imposées quant aux combinaisons de ces unités. De plus, la norme concernant le service session impose des procédures d'utilisation des primitives de service. Elle apporte ainsi des restrictions supplémentaires. Tout ceci forme un ensemble de conventions que les utilisateurs devront respecter.

Rappelons aussi que nous étudions une session orientée connexion. Ce mode de transmission définit trois phases distinctes : l'établissement de la connexion (par l'utilisation des services de connexion), la phase de déconnexion (dans laquelle on se sert des services de déconnexion), la phase de transfert située entre les deux premières (dans laquelle on utilise le reste des services).

2.1.3 Comportement général du fournisseur session

Le comportement général du fournisseur session peut être résumé en quatre règles :

- le fournisseur vérifie la disponibilité des services, l'enchaînement des primitives et la validité de leurs paramètres
- les paramètres des primitives sont fournis par l'utilisateur pour les primitives de requête et de réponse; ils sont donnés par la SPM pour les primitives d'indication et de confirmation
- à un SSDU (Session Service Data Unit) correspond un SPDU (Session Protocol Data Unit)
- les SPDU sont envoyés par la primitive de transport d'envoi de données normales

Ces règles générales sont applicables pour l'ensemble des services compris dans les différentes unités fonctionnelles. Chaque fois qu'elles ne seront pas totalement respectées, nous le soulignerons dans le texte.

Nous tenons à faire remarquer que notre intention n'est pas de décrire précisément le protocole session. Nous parlerons seulement de ses principaux mécanismes et de leur utilité. Nous essayerons aussi d'en élucider certains aspects pour le moins curieux.

2.2 L'unité fonctionnelle noyau

L'unité fonctionnelle noyau comprend les services de connexion, de déconnexion, de transfert de données normales et de fin anormale de connexion session, à savoir coupure par l'utilisateur ou par le fournisseur. Ce sont les services minimaux de session. Cette unité fonctionnelle doit toujours être proposée : sans cette unité, aucune communication de niveau session n'est réalisable.

2.2.1 Service de connexion

Le service de connexion est un service confirmé qui permet d'établir une connexion de niveau session. Lorsqu'un utilisateur désire établir une pareille connexion avec un autre utilisateur, il émet une primitive de demande de connexion. Cette primitive contient plusieurs paramètres. La référence de la connexion permet d'identifier le dialogue. Elle peut être constituée, par exemple de l'adresse de l'utilisateur complétée par la date de la connexion. L'adresse du SSAP appelant et l'adresse du SSAP appelé permettent d'établir cette connexion. La correspondance entre un SSAP et un TSAP est de type 1-1. Nous reviendrons plus en détail sur ces paramètres dans la partie pratique de notre travail. Les paramètres de qualité de service (QOS) servent à préciser quel type de connexion transport l'on désire. Viennent ensuite les propositions de l'utilisateur quant à son choix d'unités fonctionnelles et un numéro initial de série de points de synchronisation s'il désire se synchroniser sans pour autant définir d'unités logiques de travail. Les propositions de jetons liées aux différentes unités fonctionnelles détermineront les détenteurs initiaux des jetons. L'utilisateur peut aussi, éventuellement, transmettre des données.

La demande de connexion peut être soit refusée, soit acceptée. L'appelé répond par la primitive de réponse de connexion, qui comprend les mêmes paramètres, sauf le SSAP appelant, et en plus, le paramètre de résultat (acceptation ou refus).

La figure III.4 illustre un exemple d'utilisation des primitives de connexion. La référence est composée de la date et de l'heure de la demande de connexion, complétées par l'adresse de l'appelant (cette référence est identique dans la demande et la réponse). Les SSAP sont composés du nom de l'entité utilisatrice, du nom du fournisseur et d'un élément permettant de déduire le TSAP. L'appelant ne demande pas de QOS particulier, l'appelé non plus. Viennent ensuite les propositions de combinaison d'unités

fonctionnelles, la valeur du premier numéro de point de synchronisation et l'attribution des jetons. Les deux utilisateurs n'utilisent pas le paramètre données de l'utilisateur. L'appelé accepte la connexion.

requête-de-connexion (200588/1203/FNDP, présentation standard / session standard / FNDP, présentation standard / session standard / UCL, QOS standard, noyau et synchronisation mineure, 1, tous les jetons disponibles attribués à l'appelant, "pas-de-message")

réponse-de-connexion (200588/1203/FNDP, présentation standard / session standard / UCL, QOS standard, noyau et synchronisation mineure, 1, tous les jetons disponibles attribués à l'appelant, acceptation, "pas-de-message")

Fig III.4 : exemple de demande et de réponse de connexion

Rappelons que, de la négociation des paramètres, il ressortira une valeur de point de synchronisation (la valeur proposée par l'appelé), des unités fonctionnelles sélectionnées (l'intersection des deux paramètres propositions de l'utilisateur), une répartition des jetons (pour chaque jeton, l'appelant, s'il en a demandé la possession, ou si, ayant laissé le choix à l'appelé, ce dernier le lui a attribué; l'appelé dans les autres cas).

La connexion session est maintenant référençable par un identificateur de connexion CEP. On ne parle pas de cet identificateur CEP dans la norme. L'utilisation en est laissée à la discrétion des personnes désirant implémenter la norme. Remarquons qu'il ne faut pas confondre cet identificateur local de connexion avec le paramètre de référence du dialogue (identificateur de connexion). En effet, comme l'expliquent D. Day et H. Zimmermann dans un article sur le modèle de référence [DAY 83], "un identificateur de (N)-CEP est utilisé par une (N+1) et une (N) entité pour identifier une connexion particulière." L'identificateur de connexion, quant à lui, est global et transparent au fournisseur session. Il permet aux deux utilisateurs de référencer un dialogue, et donc, de le reprendre par la suite.

- Remarques sur le paramètre données de l'utilisateur

Le paramètre de données de l'utilisateur "a été un sujet important d'un meeting spécial sur l'utilisation de la session, tenu à Helsinski en septembre 1985." [OMN 86] Il peut être d'une grande utilité lors de l'établissement de la connexion. En effet, dans la philosophie actuelle de l'ISO, l'initialisation de la communication s'opère couche par couche, par ordre croissant de numéros. L'ensemble des caractéristiques de la communication est ventilé dans les divers couches, et donc, échangé petit à petit. Une couche de niveau (N) décide des caractéristiques de la connexion de niveau (N-1). Hors, nous avons vu que, pour décider des caractéristiques de la session, il faut connaître l'ensemble des caractéristiques de l'application. Dans ce modèle, c'est le niveau 6 qui décide des caractéristiques de niveau 5, alors qu'il ne connait pas toutes les caractéristiques de l'application, contenues dans les 6 et 7-PDU de connexion. Afin de laisser la décision à l'application, il serait donc intéressant "d'encapsuler" ces PDU dans le paramètre données de l'utilisateur de la primitive de demande de connexion session (figure III.5). De plus, cette technique améliorerait le temps consacré à la phase d'établissement de la communication. En effet, elle permettrait d'envoyer en une seule fois les PDU de connexion des niveaux 5, 6 et 7.

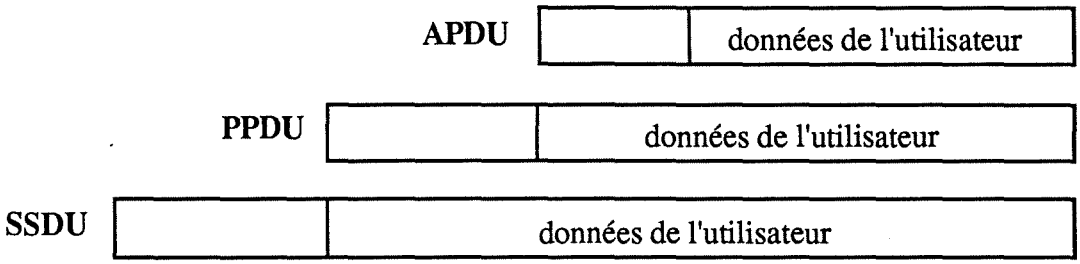


Fig III.5 : optimalisation de l'établissement de la connexion

La technique "d'encapsulation des PDU" (PDU's embedding) soulève alors le problème de la taille limitée à 512 octets du paramètre données de l'utilisateur de la primitive de connexion session. Le résultat de la discussion du meeting d'Helsinski est qu'il "fut reconnu que les niveaux 6 et 7 avaient des besoins architecturaux tels que la session ne pouvait limiter la taille de ce paramètre." [OMN 86]

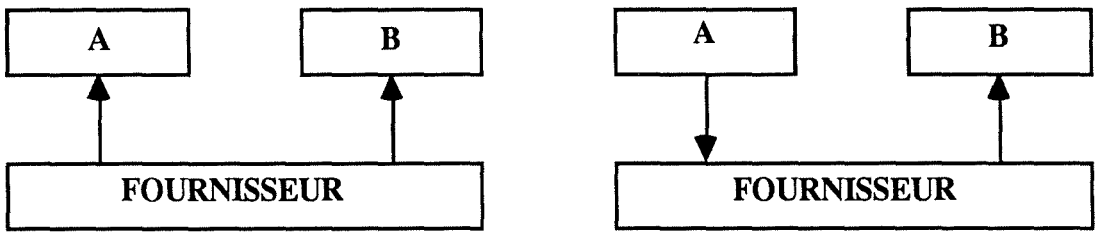
Dans la norme X410 (voir II.6.1), le RTS de MHS (Message Handling System) utilise cette technique. En effet, le paramètre données de l'utilisateur de la primitive de connexion session contient les paramètres du P-connect (primitive de connexion de niveau présentation).

2.2.2 Service d'envoi de données normales

Le service d'envoi de données normales est un service non-confirmé qui permet l'envoi de données. Il peut être soumis au contrôle du jeton de données. La taille de ces données est illimitée. Notons que les primitives d'envoi de données normales transportent aussi bien des informations de contrôle que des données (au sens commun du terme) de niveau supérieur.

2.2.3 Service de fin anormale de session

Le service de fin anormale de session permet de réaliser une coupure "brutale" de la connexion session : les utilisateurs perdent leur connexion session, et éventuellement ce qui se trouvait dessus à cet instant. Une fin anormale de session est signalée, soit à un utilisateur par une primitive de coupure émise par l'autre utilisateur (partie droite de la fig. III.6), soit aux deux utilisateurs par deux primitives d'indication de coupure émises par le fournisseur (partie gauche de la fig. III.6).



coupure fournisseur *coupure utilisateur*

Fig III.6 : coupure de la session

La primitive de coupure par l'utilisateur du service session comprend comme seul paramètre un champ données de l'utilisateur, limité à 9 octets. Cette limitation nous parait, ainsi qu'aux concepteurs d'ISODE [ROS 87], très restrictive quant à son utilisation possible par les niveaux supérieurs. Si ce champ avait été plus long, il aurait permis de donner la raison complète de la coupure ainsi que, par exemple, une trace des événements entraînant cette décision.

La primitive de coupure par le fournisseur contient un paramètre de raison. Nous reviendrons sur ces primitives lors du point explicitant le traitement des erreurs.

2.2.4 Service de déconnexion

Le service de déconnexion est un service qui permet de couper la connexion session sans perte de données, par demande-confirmation.

L'utilisateur possédant l'ensemble des jetons disponibles a seul le droit de demander la déconnexion. Si aucun jeton n'est disponible, les deux utilisateurs peuvent la demander. Celui qui reçoit la demande n'a pas le droit de refuser (voir aussi III.2.7).

Les primitives de demande et d'indication de déconnexion contiennent comme paramètre un champ de données utilisateur. Les primitives de réponse et de confirmation contiennent, en plus, le paramètre de résultat. Ce paramètre n'est ici d'aucune utilité.

2.2.5 Remarques sur le fournisseur

En ce qui concerne la phase de connexion, comme le fait remarquer Fausto Caneschi dans son article [CAN 86] : à une requête correspond un seul SPDU (connect), et à une réponse correspond soit le SPDU "accept" soit le SPDU "refuse". "L'idée des concepteurs de la norme était de différencier fortement syntaxiquement ce qui l'était sémantiquement (oui opposé à non)". [CAN 86]

Le fournisseur du service session peut décider de garder ou de libérer la connexion transport, lors de la terminaison normale ou anormale de la connexion session. Il pourra par la suite décider de réutiliser une connexion transport non libérée, sous certaines conditions. De notre analyse, il est apparu que ces choix ne sont pas du ressort des utilisateurs. En effet, aucun paramètre de primitive ne permet à un utilisateur de signifier au fournisseur qu'il désire garder la connexion.

Lors de l'échange des SPDU de connexion, les deux entités session correspondantes s'échangent des tailles de TSDU (Transport Service Data Unit) maximales, pour les deux sens du transfert. La plus petite taille sera conservée pour chacun des sens de transfert. Ceci peut donc engendrer des mécanismes de segmentation (1 SSDU => n SPDU). Ce mécanisme est réalisé à l'intérieur du fournisseur sans que les utilisateurs en soient avertis.

La concaténation étendue est demandée par un paramètre QOS. Ce mécanisme permet de rassembler plusieurs SPDU de types différents (correspondant à des demandes de service différentes) dans un TSDU. Ceci permet d'utiliser de façon optimale, le service d'envoi de données de niveau transport. La concaténation de base, elle, est toujours réalisée. Son objectif est très différent : elle permet uniquement de réaliser la correspondance avec le codage des SPDU tel que défini dans la norme ISO et celui de T62 (télétex). C'est ainsi qu'un SPDU d'envoi de données est toujours concaténé avec un SPDU de passation de jetons, même si ce dernier ne correspond à aucune demande de service. Pour plus de détails sur la concaténation, le lecteur pourra se référer à la norme.

Le contrôle étendu est exercé par le fournisseur du service session, s'il est demandé par l'intermédiaire d'un paramètre QOS, ou si le flux express est demandé et accordé. Il permet l'envoi des SPDU coupure et acceptation de coupure sur le flux express. De plus, il permet, lors de l'émission de certaines primitives, l'envoi d'un SPDU 'prepare' sur ce même flux. La réception de ce SPDU peut entraîner l'abandon ou la retenue, par la SPM, de certains SPDU, envoyés en même temps ou ultérieurement. Voici les SSDU qui impliquent l'envoi du SPDU 'prepare' :

- resynchronisation (demande et réponse)
- interruption et abandon d'activité (demande et réponse)
- synchronisation majeure et terminaison d'activité (réponse).

L'avantage de ce contrôle est qu'il permet, grâce au flux express, de détecter plus rapidement une anomalie, et ainsi de bloquer et de débloquer au plus tôt la SPM. Il permet aussi de libérer au plus tôt une connexion lors d'une coupure.

Prenons l'exemple de la resynchronisation. Suite à une erreur (peu importe laquelle), un utilisateur décide de resynchroniser. Sa demande de resynchronisation entraînera l'envoi par sa SPM du SPDU "préparation de resynchronisation" sur le flux express. Ce

SPDU permettra de bloquer au plus tôt la SPM de son correspondant : tout SPDU arrivant après la demande de préparation sera abandonné (il n'y correspondra pas d'indication ou de confirmation). Le SPDU "prepare" envoyé suite à la réponse de resynchronisation permettra lui de débloquer au plus tôt la SPM du demandeur de la resynchronisation.

2.3 L'unité fonctionnelle de terminaison négociée

Dans notre exposé de l'unité fonctionnelle noyau, nous avons vu que, lors de la réception d'une primitive indication de déconnexion, l'utilisateur ne peut qu'accepter cette demande. Lorsque l'unité fonctionnelle de terminaison négociée est retenue (en plus du noyau), l'utilisateur a le droit de refuser cette demande (voir aussi III.2.7).

Rappelons que cette unité implique l'existence du jeton de terminaison négociée. Cette unité fonctionnelle regroupe donc , en plus du service de terminaison normale, les services non-confirmés de gestion de jeton (demande et cession de jetons).

En ce qui concerne le fournisseur, notons que, tout comme lors de la connexion, à une requête correspond un SPDU (finish), et à une réponse correspond soit le SPDU "disconnect" soit "non-finished". La justification proposée par Fausto Caneschi concernant les SPDU de connexion est également applicable ici (voir III.2.2.5).

2.4 Les unités fonctionnelles half et full duplex

L'unité fonctionnelle half duplex permet de réaliser un transfert de données normales dans un seul sens à la fois (échange de données dans un seul sens ou dans les deux sens à l'alternat). Elle est caractérisée par la disponibilité du jeton de données. L'unité fonctionnelle full duplex permet de réaliser un transfert dans le deux sens à la fois. Le jeton de données n'est alors pas disponible. Ces deux unités ne peuvent donc être adoptées simultanément pour une même connexion.

Le service d'envoi de données normales est contenu dans le noyau. L'unité half duplex ne regroupe donc que les services de gestion de jetons. L'unité full duplex ne regroupe, elle, aucun service.

L'adoption de ces deux unités fonctionnelles implique l'attribution ou non du jeton de données. La disponibilité de ce jeton a des influences sur l'utilisation d'autres services. Nous les citerons par la suite.

Ainsi que nous le verrons au point III.2.5, il existe d'autres possibilités d'envoyer des données. Malgré l'adoption de l'unité half duplex, l'utilisateur ne disposant pas du jeton peut donc quand même, dans certains cas, en envoyer.

2.5 Envoi de données

Par l'intermédiaire des services session, il existe un grand nombre de façons d'envoyer des données. A des fins de clarté, nous nous sommes permis de les rassembler dans ce point sous le label "envoi de données".

Nous verrons, dans un premier point, les unités fonctionnelles spécialement conçues pour l'envoi de données. Pour ce faire, nous prendrons comme point de comparaison les caractéristiques complètes de l'envoi de données normales (unité fonctionnelle noyau). Nous verrons, dans un deuxième point, les autres possibilités.

2.5.1 Les unités fonctionnelles d'envoi de données

Nous avons déjà vu un service d'envoi de données : le service d'envoi de données normales contenu dans l'unité fonctionnelle noyau. Voici maintenant les unités fonctionnelles destinées à l'envoi de données :

- l'unité fonctionnelle de données typées
- l'unité fonctionnelle de données expresses
- l'unité fonctionnelle d'information de capacité.

Présentons d'abord l'ensemble des caractéristiques de l'envoi de données normales. C'est un envoi non-confirmé sur le flux des données normales. Il est toujours disponible et est soumis à la disponibilité d'un jeton. Il peut être structuré (synchronisation). La taille des données normales est illimitée. Les données peuvent être perdues suite à l'utilisation de tout mécanisme de traitement d'erreur (voir traitement d'erreur). L'envoi est pris en compte pour le calcul du débit (paramètre QOS). Il est toujours concaténé et peut subir la segmentation. Ce flux correspond au flux de données habituellement disponible. Il sert à l'envoi de données au sens commun du terme.

Le flux de données typées possède des caractéristiques très semblables au flux de données normales. La seule grosse différence est qu'il ne subit pas de contrôle half duplex. Ce service permet donc le transfert de données pseudo-normales indépendamment du jeton de données normales. Cette unité fonctionnelle peut être demandée dans le cas d'un échange half duplex de données, tout en gardant la possibilité de réaliser, sans contrainte, un dialogue de contrôle de niveau application. Nous reprendrons, comme exemple

d'utilisation de cette unité fonctionnelle dans les autres normes, l'emploi fait de ce service dans le CASE CCR (Common Application Service Element Commitment Concurrency Recovery). La primitive de service correspondante permet de véhiculer la réponse de l'esclave, consécutive à la demande d'une action du maître (véhiculée par la primitive de données normales), lorsque l'action a été réalisée par l'esclave (figure III.7).

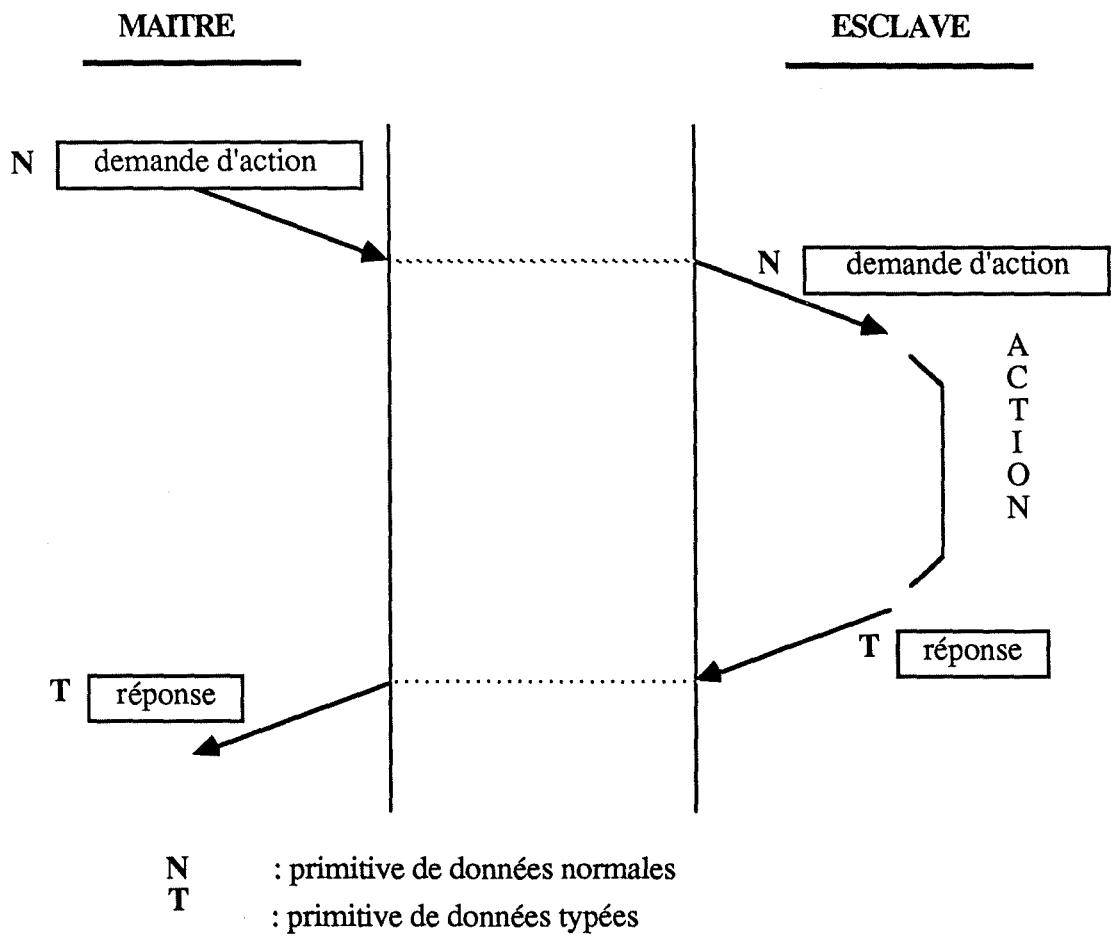


Fig III.7 : réalisation d'une action d'après le CCR

Le flux de données expresses se différencie largement des deux flux précédents. Ses deux caractéristiques principales sont : l'emploi du flux express transport, la taille très limitée des informations. Nous entrevoyons, comme utilisation de ce flux, l'envoi limité et urgent de données de contrôle de niveau supérieur. Par exemple, dans la couche session, le fournisseur se sert du flux express transport pour envoyer certains SPDU (coupure, acceptation de coupure). Il ne faut pas oublier que la sélection de ce flux implique obligatoirement la réalisation, par le fournisseur session, du contrôle étendu (III.2.2.5).

Le flux d'informations de capacité se différencie, lui aussi largement, du flux de données normales. Sa caractéristique principale résulte du lien profond qui existe entre cette unité fonctionnelle et l'unité fonctionnelle de gestion d'activité. L'unité fonctionnelle d'information de capacité ne peut être proposée qu'avec l'unité fonctionnelle de gestion d'activité. Ainsi que nous le verrons par la suite, c'est le seul moyen d'envoyer de l'information à son vis-à-vis, de façon synchronisée, en dehors d'une activité. En effet, c'est un service confirmé qui est bloqué tant que la dernière primitive de même type n'a pas été acquittée (figure III.8). Une autre caractéristique importante est le rapport maître-esclave, imposé par l'existence de jetons. Il faut remarquer que, dans la primitive d'acquittement, de l'information, provenant de l'utilisateur esclave, peut être insérée.

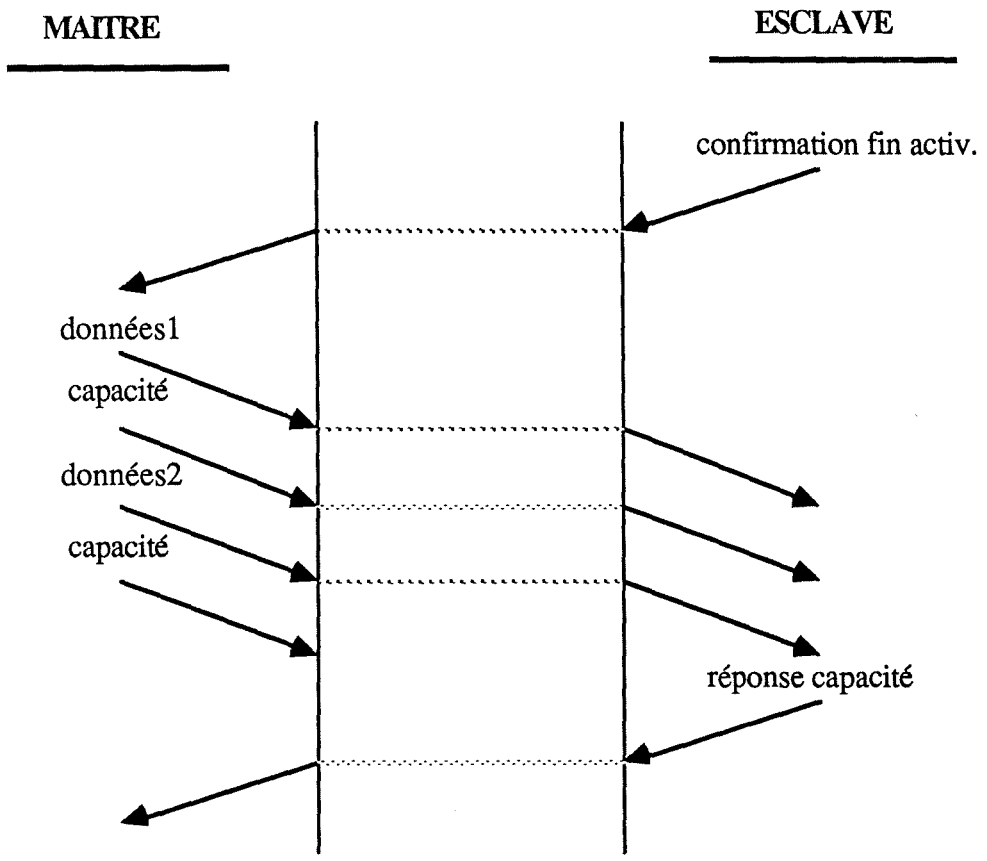


Fig III.8 : échange d'information de capacité

Cette unité fonctionnelle peut donc être utilisée afin de transférer de l'information de contrôle de niveau 6 ou 7, devant être confirmée, et ceci en dehors d'une unité logique de travail. Pour illustrer l'utilisation de cette primitive, nous nous référons à la norme T62. Les éléments de protocole, relatifs aux primitives d'envoi et d'acquittement d'informations de capacité, sont repris exactement de la norme T62, sous la forme CDCL et RDCL (Command et Response Document Capability List). Ces éléments permettent, en dehors des

liens de documents (notion apparente d'activité), d'échanger des informations concernant la capacité des terminaux télétex, à ce moment précis du dialogue.

CONTROLE	NORMALE	TYPEE	EXPRESSE	CAPACITE
confirmation	non-confirmé	non-confirmé	non-confirmé	confirmé
perte	possible	possible	possible	possible
flux	normal	normal	express	normal
débit	influence	influence	pas d'infl.	pas d'infl.
jetons données	indisp/attrib	—	—	indisp/attrib
jetons mineur	—	—	—	indisp/attrib
jetons majeur	—	—	—	disponible
jetons terminais.	—	—	—	—
STRUCTURE				
majeur	oui	oui	oui	non
mineur	oui	oui	non	non
AUTRES				
disponibilité	toujours	au choix	au choix	au choix
taille	illimitée	illimitée	1 .. 14 octets	512 octets
concaténation	toujours	jamais	jamais	toujours
segmentation	possible	possible	jamais	possible

__ : aucune restriction pour l'envoi de données correspondant

Tableau III.1 : caractéristiques des envois de données

Le tableau III.1 est un tableau synoptique reprenant les diverses caractéristiques des services spécialisés d'envoi de données. Il met en correspondance leurs différentes particularités.

2.5.2 Autres possibilités

Il est aussi possible de transférer 512 octets d'information par l'intermédiaire de l'ensemble des services session, hormis :

- la cession de jetons et la passation de contrôle
- l'interruption et l'abandon d'activité
- la coupure et la signalisation d'anomalie, par le fournisseur du service session.

L'information est véhiculée dans le paramètre "données de l'utilisateur" des primitives de service correspondantes.

Nous avons déjà vu une utilisation du paramètre données de l'utilisateur pour les primitives d'ouverture de connexion. Nous en exposerons par la suite des exemples d'utilisation pour d'autres primitives.

2.6 Synchronisation majeure et mineure

Afin de souligner les différences qui existent entre les services de synchronisation mineure et majeure, nous nous sommes permis de regrouper les exposés des unités fonctionnelles correspondantes sous le label "synchronisation mineure et majeure".

Nous exposerons, en un premier temps, les unités fonctionnelles de synchronisation mineure et majeure. Ensuite, afin de mieux comprendre leurs caractéristiques, nous ferons une comparaison entre ces deux unités.

2.6.1 L'unité fonctionnelle de synchronisation mineure

L'unité fonctionnelle de pose de points de synchronisation mineurs comprend les services de pose et d'acquittement de points de synchronisation mineurs. De plus, elle comprend les services de gestion des jetons (demande et cession).

La primitive de pose de point de synchronisation mineur contient les paramètres suivants : type, numéro de série et données de l'utilisateur. Le numéro de série est renvoyé à l'utilisateur par le fournisseur du service session. La série commence au numéro de base (fourni à l'initialisation de la connexion ou au début d'une activité). Ce numéro est incrémenté de 1 à chaque demande. Les autres paramètres sont, comme d'habitude, fournis à la SPM. Le paramètre "type" permet au maître d'informer l'esclave quant à son désir de recevoir la confirmation des points émis. Il prend la valeur explicite ou optionnelle. Cette valeur ne concerne que les utilisateurs du service.

La confirmation d'une pose de point de synchronisation mineur est peu contrôlée par la SPM. Elle doit être gérée par les utilisateurs. La SPM ne vérifie que le numéro fourni lors de la confirmation : il doit correspondre à un point de synchronisation non encore confirmé. Il est à noter que la confirmation d'un point de synchronisation acquitte aussi les points de synchronisation mineurs précédents non encore confirmés.

Rappelons que ce service est lié à la présence de jetons, et que son utilisation est de nature exclusive. Afin de poser un point de synchronisation, on doit disposer du jeton de synchronisation mineure et de celui de données s'il est disponible.

Nous prendrons, comme exemple dans les normes, la gestion des mineurs exercée dans X410, qui est, dailleurs, très semblable à celle définie dans la norme T62. Dans

X410, on définit une taille de fenêtre, qui détermine le nombre de points de synchronisation mineurs pouvant être en attente de confirmation, avant que le transfert de données ne soit suspendu. La confirmation a été définie comme obligatoire. Pour T62 par exemple, la taille standard de la fenêtre est de 3.

2.6.2 L'unité fonctionnelle de synchronisation majeure

L'unité fonctionnelle de synchronisation majeure comprend les services de pose et de confirmation de pose de points de synchronisation majeurs. Elle comprend aussi les services de gestion de jetons.

La primitive de pose de point de synchronisation majeur possède, comme paramètre, le numéro de série fourni par la SPM ainsi qu'un champ de données de l'utilisateur. Cette primitive doit être obligatoirement confirmée à sa réception. La SPM bloquera toute demande (sauf celles de traitement d'erreurs) jusqu'à confirmation de ce point. La confirmation d'un point de synchronisation majeur confirme tous les points de synchronisation mineurs non encore acquittés.

Rappelons que ce mécanisme est lié à l'existence de jetons. Afin de poser un point de synchronisation majeur, on doit disposer du jeton majeur, du jeton mineur et du jeton de données s'ils sont disponibles.

Remarquons que, en ce qui concerne le fournisseur, seul l'acquittement d'un point majeur entraîne l'émission d'un SPDU 'prepare' (III.2.2.5). Il sert à débloquer au plus tôt la SPM appelante. L'envoi de pareil SPDU, lors de la pose de point majeur, ne serait d'aucune utilité.

2.6.3 Comparaison

Avant de rentrer dans le vif du sujet, notons l'indépendance réelle qui existe entre ces deux unités fonctionnelles. Il est possible de structurer le dialogue uniquement par la synchronisation mineure. Le dialogue ne serait alors composé que d'une unité de dialogue découpée en sous-unités.

L'indépendance au niveau de l'utilisation des services est plus restreinte. Analysons pour cela les relations entre ces services et les différents jetons. Ces deux mécanismes de pose de points de synchronisation sont soumis au contrôle de jetons. Pour poser des points mineurs ou majeurs, on doit disposer du jeton de données, s'il est disponible. Notons que la pose de points majeurs nécessite la possession du jeton mineur. Par contre, la pose de points mineurs ne tient pas compte du jeton majeur. Ceci signifie que, à un moment donné, les détenteurs des jetons de synchronisation peuvent être différents. Toutefois, nous considérons que, afin de respecter les caractéristiques de la synchronisation mineure (pas de pause dans le dialogue), en half duplex, les jetons mineur et de données doivent être détenus par un même utilisateur. Dans le cas contraire, la pose d'un point mineur devrait être précédée d'un échange de jeton, et donc, d'un arrêt du dialogue.

Ces deux mécanismes de structuration n'interviennent que sur un sens de transfert à la fois. Ils sont de nature half duplex ou maître-esclave. En full duplex, il est impossible vu les mécanismes proposés, de structurer les deux sens de transfert simultanément. C'est pour cela que les concepteurs de la norme ont proposé un addendum (la synchronisation symétrique), permettant de poser des points de synchronisation majeurs sur les deux sens de transfert à la fois, de gérer ainsi deux séries de numéros et donc de se resynchroniser en un point précis de chacun de ces sens (figure III.9). Notons que, lorsqu'on sélectionne la synchronisation symétrique, on ne peut se servir que de la synchronisation majeure comme mécanisme de structuration.

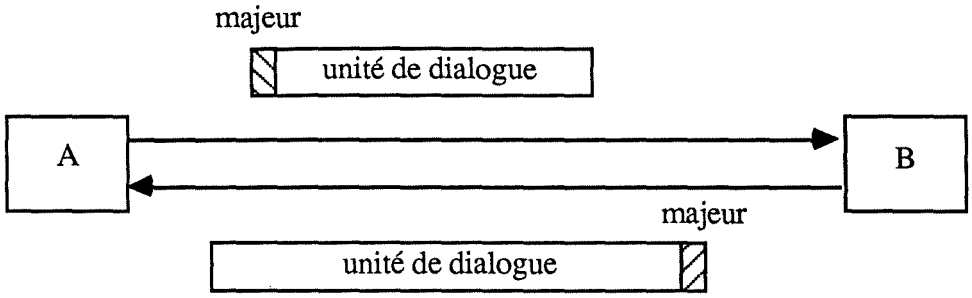


Fig III.9 : synchronisation symétrique

Le contrôle exercé par la session sur les points majeurs est plus important que celui exercé sur les points mineurs. Un majeur doit être confirmé et bloque le dialogue jusqu'à sa confirmation. C'est de cette manière que ce repère sépare donc clairement les unités de dialogue. La gestion de la confirmation des mineurs est laissée aux soins des utilisateurs du service.

Notons que, pour la SPM, la confirmation d'un point majeur est explicite (obligation de répondre explicitement à ce point) et non-transparente (obligation contrôlée par la SPM). La confirmation d'un mineur est, elle, transparente et peut être implicite. En effet, un point de synchronisation mineur posé par un utilisateur A est considéré comme confirmé par la SPM, lors de la réception de la réponse de l'utilisateur B à un point mineur suivant ou à un point majeur. Lorsque la pose du point mineur par l'utilisateur A a été suivie d'un échange de jetons, il est aussi considéré comme confirmé par la réception d'une demande de pose de point mineur ou majeur provenant alors de l'utilisateur B. Ceci est à mettre en rapport avec l'unicité de la série de points de synchronisation.

Le choix d'un de ces mécanismes de structuration, et la gestion qui peut en suivre, influencent les performances de l'échange. Imaginons simplement un transfert de pages délimitées soit par un majeur, soit par un mineur. La figure III.10 illustre ce transfert dans le temps. En ce qui concerne l'emploi du point mineur, on suppose, dans cette illustration, que la taille de fenêtre est très grande. Cela signifie que l'émetteur peut poser un très grand nombre de points mineurs avant de s'arrêter pour en attendre la confirmation (si celle-ci n'est pas arrivée entre temps). Les conséquences sur les performances de l'échange sont évidentes : d'un côté (synchronisation majeure) l'émetteur envoi une page et ensuite s'arrête jusqu'à ce qu'il ai reçu la confirmation de son correspondant; de l'autre (synchronisation mineure), l'émetteur envoie autant de pages que sa fenêtre le lui permet et ne s'arrêtera que si la confirmation des pages n'est pas arrivée entre temps.

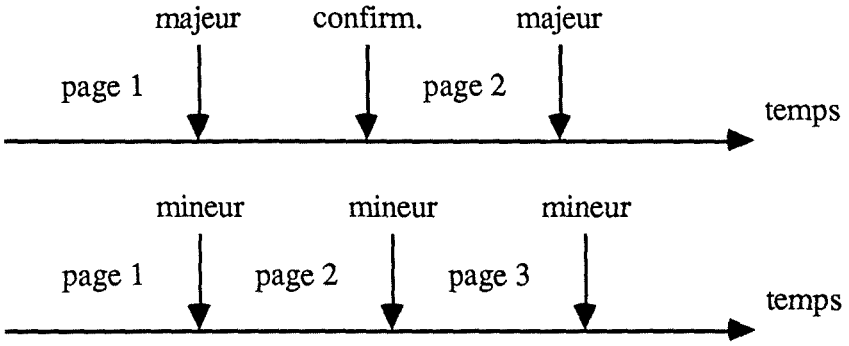


Fig III.10 : transfert de pages dans le temps

2.7 L'unité fonctionnelle de resynchronisation

L'unité fonctionnnelle de resynchronisation comprend le service de resynchronisation. Ce service est de type confirmé.

Les primitives de resynchronisation comprennent les paramètres "type", "numéro de série", "attribution des jetons" et "données de l'utilisateur".

Le paramètre "type", présent uniquement dans la requête et l'indication, indique à quel type de resynchronisation on est confronté. Rappelons les différentes options : redémarrage, abandon, choix de l'utilisateur.

Le paramètre "attribution des jetons" contient une proposition de même type que celle exposée lors de la connexion et véhicule ainsi une proposition de nouvelle affectation de ces derniers. L'utilisation de la primitive de resynchronisation, qui n'est d'ailleurs soumise à aucun jeton, peut être un moyen de s'en emparer.

Le prochain numéro de série de synchronisation sera établi de façon différente, d'après l'option de resynchronisation désirée. Dans l'option abandon , la primitive de requête n'a pas besoin de numéro : il est fourni par la SPM expéditrice, et sera indiqué à l'utilisateur récepteur qui acquittera avec ce numéro. Lors de la confirmation, la SPM et l'utilisateur qui étaient à la base de la resynchronisation, seront informés de cette valeur. Cette valeur est, rappelons-le, une valeur jamais encore utilisée. Dans l'option redémarrage, le demandeur va imposer son numéro au receveur, qui en sera informé lors de l'indication. Dans l'option choix de l'utilisateur, le demandeur propose un numéro au receveur, qui décidera du numéro final. Le demandeur en sera informé à la confirmation. C'est ainsi qu'il est possible de repartir à zéro, de revenir en arrière dans le dialogue, ou de reprendre uniquement dans l'unité courante.

Il est à noter qu'après une resynchronisation avec option abandon ou choix de l'utilisateur, la possibilité est offerte de redémarrer à partir du début du dialogue. Après une resynchronisation option redémarrage, la possibilité de redémarrer à nouveau est toujours limitée à la même unité de dialogue. Si la procédure paraît logique pour les options redémarrage et abandon, elle l'est beaucoup moins quant à l'option choix de l'utilisateur. En effet, la SPM permet, suite à une resynchronisation option choix de l'utilisateur, de redémarrer à partir de la première unité de dialogue, au lieu de se limiter à l'unité courante (figure III.11).

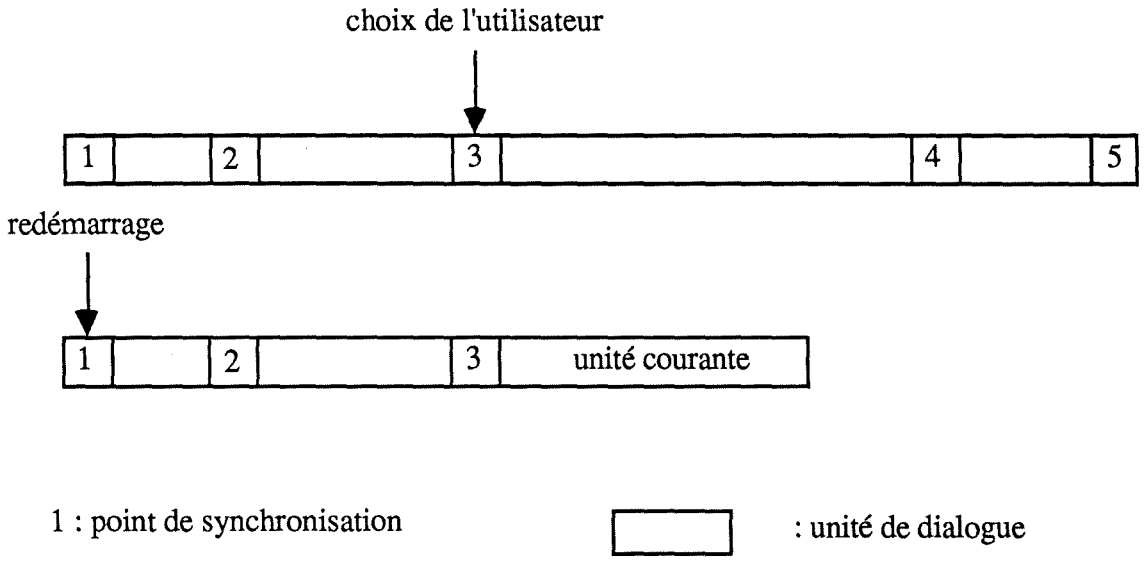


Fig III.11 : combinaison de resynchronisations

Nous en concluons que, surtout par l'option abandon, mais aussi par ses autres options, la resynchronisation permet de remettre en cause l'intégrité des unités de dialogue confirmées. Ceci nuance la définition de la norme quant à la complète séparation des unités de dialogue. Voilà pourquoi nous avons préféré employer l'expression "clairement séparées" à propos des unités de dialogue, au moment où nous avons parlé des unités fonctionnelles de synchronisation.

Abordons maintenant l'indépendance des unités fonctionnelles de synchronisation et de resynchronisation. Cette indépendance est réelle : aucune restriction de combinaison de ces unités n'est reprise dans la norme. Cela veut dire qu'un utilisateur, dans sa combinaison d'unités fonctionnelles, peut reprendre la resynchronisation, sans pour autant retenir les unités fonctionnelles de synchronisation. Néanmoins, lorsque seule l'unité de resynchronisation est sélectionnée, étant donné qu'il est impossible de poser des points de repère sur le dialogue, on ne peut resynchroniser qu'avec option abandon. Ceci est vérifié par la SPM. L'indépendance de ces unités fonctionnelles permet de garder un mécanisme de prise de jetons et de traitement d'erreur, lorsque la structuration en unités ou sous-unités de dialogue n'est pas sélectionnée par les applications.

Nous tenons à faire remarquer une utilisation particulière de la resynchronisation proposée dans la norme. Il est permis, lors de la réception d'une demande de terminaison normale de connexion, si l'unité fonctionnelle de gestion d'activité n'a pas été sélectionnée, de la refuser en émettant une demande de resynchronisation. Ceci nuance fortement l'explication de la procédure de déconnexion donnée aux points III.2.2.4 et III.2.3, dans la mesure où nous avions laissé sous-entendre que, si on ne sélectionnait pas l'unité fonctionnelle de terminaison négociée, il n'était pas possible de refuser une demande de déconnexion.

2.8 L'unité fonctionnelle de signalisation d'anomalie

L'unité fonctionnelle de signalisation d'anomalie comprend les services de signalisation d'anomalie par l'utilisateur, et par le fournisseur du service session. Les primitives correspondantes comprennent un paramètre de raison. Seule la primitive demandable par l'utilisateur contient un champ de données.

Une fois une anomalie détectée, la SPM passe en état d'anomalie, d'où elle ne sortira que grâce aux primitives de traitement d'erreur. Aucun autre service ne peut être demandé lorsque le fournisseur se trouve dans cet état.

Contrairement à ce que pourrait laisser sous-entendre la partie service de la norme, ces deux mécanismes sont liés à l'existence du jeton de données : seuls l'utilisateur et la SPM qui ne disposent pas du jeton de données ont le droit d'utiliser cette signalisation. Notons que ce jeton doit alors obligatoirement être disponible. Ceci implique que l'unité de signalisation d'anomalie ne peut être proposée qu'avec l'unité half duplex.

La restriction imposée par les concepteurs de la norme sur l'utilisation de ce mécanisme peut paraître absurde : le propre de ces mécanismes est d'être utiles aux deux utilisateurs et au fournisseur. [CAN 86] Nous avons trouvé la raison de cette restriction dans cet article. Elle provient du souci des concepteurs de la norme de la rendre compatible avec la norme T62, ce qui a entraîné l'intégration d'un nouveau service (la signalisation d'anomalie), alors que le protocole était lui déjà presque stable.

2.9 L'unité fonctionnelle de gestion d'activité

L'unité fonctionnelle de gestion d'activité comprend les services de gestion d'activité et de passation de contrôle. Nous présenterons cette unité fonctionnelle dans cet ordre. Ensuite, nous comparerons les caractéristiques des deux principaux outils de structuration, à savoir : l'activité et l'unité de dialogue. Nous en donnerons alors des exemples d'utilisation.

2.9.1 Services de gestion d'activité

Nous allons d'abord présenter les services attachés à la gestion d'activité en les séparant en trois groupes correspondant à trois phases distinctes : le démarrage, la fin normale et la terminaison anormale d'activité. Après cela, nous exposerons les principales retombées des procédures d'utilisation de ces services sur les concepts de la couche session.

2.9.1.1 Présentation des services

Les services de gestion d'activité sont les services de démarrage, de fin normale et de fin anormale d'activité. Seuls les services de démarrage ne sont pas confirmés. L'ensemble est soumis au jeton majeur et d'activité. Seuls les services de fin anormale ne sont pas soumis aux jetons mineur et de données.

- Services de démarrage d'activité

Les primitives de lancement et de reprise d'activité signalent le "démarrage" d'une unité logique de travail identifiée. Les primitives correspondantes contiennent comme paramètres communs : un identificateur d'activité transparent au service session, et des données de l'utilisateur. La primitive de reprise comprend, en plus, des paramètres relatifs à l'ancienne connexion (identificateur de l'ancienne connexion, ancien identificateur d'activité), afin d'établir le lien de reprise. Elle possède aussi un champ contenant un numéro de série de synchronisation. Ce numéro ne subit aucune restriction. Il sera imposé à l'utilisateur récepteur de la demande. Lors d'un lancement simple d'activité, le numéro de

série est initialisé à 1. IL correspond donc, à chaque activité, une série de points de synchronisation.

- Service de fin normale d'activité

Au service de fin anormale d'activité correspondent les primitives de terminaison d'activité. Celles-ci contiennent comme paramètres : des données de l'utilisateur et un numéro de synchronisation. Ces primitives servent à la fois de clôture d'unité logique de travail et de pose de point de synchronisation majeur.

Ce mécanisme de terminaison d'activité est très semblable à celui de pose de point de synchronisation majeur. Les SPDU correspondants ont le même type (identificateur de SPDU), et ne se différencient que de peu. Les accusés de réception sont les mêmes. Ceci implique l'existence de variables d'état, dans les SPM, permettant de les différencier.

- Services de fin anormale d'activité

Aux services de fin anormale d'activité correspondent les primitives d'interruption et d'abandon d'activité. Elles clôturent anormalement l'unité logique de travail. Elles comprennent comme paramètre la raison de la terminaison anormale (par exemple : capacité de réception compromise, erreur de procédure irrémédiable).

La norme précise les retombées attendues sur le dialogue par l'emploi de ces mécanismes. Un abandon d'activité sous-entend que son contenu est annulé. Une interruption permet, quant à elle, de signifier que l'activité sera reprise par la suite.

Il est à noter que ces mécanismes ne sont pas, à proprement parler, des mécanismes de structuration, mais bien de traitement d'exception. Remarquons que le demandeur de fin anormale d'activité n'est soumis qu'à la possession du jeton majeur et d'activité. Il recevra, lors de la confirmation, l'ensemble des jetons disponibles, et donc le contrôle de la communication. Ceci lui permet d'avoir en sa possession tous les éléments nécessaires pour réagir face à un état d'exception

2.9.1.2 Retombées sur les concepts session

Lorsque l'on termine une activité, on pose implicitement un point de synchronisation majeur. Cela signifie que, si l'unité logique de travail était structurée en unités de dialogue, il n'est pas nécessaire de poser un dernier point de synchronisation majeur confirmant la dernière unité de dialogue, avant de terminer l'activité. Même si l'unité logique de travail n'est pas découpée en unités de dialogue, la fin normale d'une activité détermine la fin d'une unité de dialogue. Ceci renforce l'hypothèse selon laquelle une unité logique de travail est composée d'au moins une unité de dialogue.

Une seule activité, au maximum, se déroule, à un moment donné, sur une connexion session. Une série de points de synchronisation lui est associée, cette série commençant à 1. Cela signifie qu'il est impossible, dans le cadre d'une activité, de se resynchroniser à une unité de dialogue n'appartenant pas à cette unité logique de travail. C'est pour cette raison, à notre avis, que les unités de dialogue appartenant à une activité sont complètement séparées des autres unités de dialogue. Nous reviendrons sur ce point lors de la comparaison entre ces deux outils de structuration.

Le contrôle supplémentaire exercé par le fournisseur du service session est, a priori, assez limité. En effet, celui-ci contrôle simplement le fait que l'on soit en activité ou non. Cela revient à dire, par exemple, que l'on ne peut lancer une activité que s'il n'y a pas d'activité en cours, et réciproquement, que l'on ne peut en terminer une que si une activité est en cours. La session n'a pas de main-mise sur l'identificateur d'activité : c'est un paramètre transparent au fournisseur du service session. Cela signifie que, en théorie, rien n'est prévu pour empêcher un utilisateur de reprendre une activité qu'il n'a jamais commencée.

Ce contrôle, a priori limité, s'inscrit pleinement dans la démarche suivie par les concepteurs de la norme ISO. Rappelons que la session se veut offrir un instrument syntaxique d'organisation et de structuration du dialogue, et en contrôler la procédure d'utilisation. La valeur sémantique attachée à la structuration n'est connue que de l'application. Toute signification d'un point de repère (simple début d'une unité logique de travail, début signifiant reprise d'une unité logique de travail), est volontairement laissée à la discrètion des niveaux supérieurs.

De plus, lorsqu'on abandonne une activité, le fournisseur ne fait que signifier à l'autre utilisateur que le contenu de l'activité doit être abandonné. Vu le mécanisme adopté,

les SPM ne gardent pas les données qui leur sont confiées, pour ne les livrer qu'à la fin d'une activité. Une fois les données délivrées, elles ne possèdent plus de moyens d'action sur elles, et de ce fait, elles ne peuvent elles-même abandonner cette partie du dialogue.

L'adoption de l'unité fonctionnelle d'activité a d'énormes implications sur l'utilisation des autres services. Lorsque l'unité fonctionnelle d'activité est adoptée, ne peuvent s'exercer :

- qu'à l'intérieur d'une activité, les mécanismes de :
  - synchronisation majeur et mineur
  - resynchronisation
  - signalisation d'anomalie
- qu'à l'extérieur d'une activité, les mécanismes de :
  - passage de contrôle
  - envoi de données de capacité.

Cela siginifie que, en dehors d'une activité, il est possible d'envoyer et recevoir des données, mais qu'il est impossible de les structurer et de se resynchroniser sur cette partie du dialogue. L'envoi de données de capacité est alors le seul moyen de structurer et de synchroniser un échange de données. Cela signifie en plus, ainsi que nous le verrons plus loin, que toute erreur de protocole dans cette partie du dialogue pourrait avoir des conséquences graves, comme la coupure de la connexion (III.3.3.1).

Remarquons enfin que, lorsque l'unité fonctionnelle a été adoptée lors de la demande de connexion, il ne sert à rien de passer de valeur de point de synchronisation. Le numéro de début de série est initialisé lors de l'ouverture (valeur 1) ou lors de la reprise (valeur proposée par l'utilisateur) d'une activité.

2.9.2 Service de passation de contrôle

Les primitives correspondant au service de passation de contrôle ne possèdent aucun paramètre. Elles permettent de transmettre l'ensemble des jetons, de celui qui les possède à l'autre utilisateur, et ceci en dehors d'une activité.

Il est à noter, premièrement, que ce service peut être réalisé par la primitive habituelle de passation de jetons et, deuxièmement, que ce service (pourtant non-confirmé) est confirmé au niveau du protocole session. Il provient, en fait, des contraintes de compatibilité avec T62. [CAN 86]

2.9.3 Comparaison activité - unité de dialogue

Une façon de voir ce qu'apporte la notion d'activité vis-à-vis de l'unité de dialogue, est d'examiner si les caractéristiques d'une activité peuvent être simulées, uniquement par la synchronisation majeure (tableau III.2).

CRITERES	ACTIVITE	UNITE DE DIALOGUE
une seule à la fois	oui	oui
abandon	oui	oui
prolongement sur une autre connexion	oui	possible
interruption et reprise : même connexion	oui	possible
autre connexion	oui	possible
identification	oui	possible

Tableau III.2 : comparaison activité - unité de dialogue

Les deux premières caractéristiques (une seule à la fois sur une connexion, possibilité d'abandon) sont communes. Afin de réaliser les autres (prolongement sur une autre connexion, interruption et reprise, identification), il est nécessaire d'adopter plusieurs conventions. Une valeur sémantique particulière doit être attachée à certains points de synchronisation, à savoir : début, reprise, interruption et fin d'activité. Notons que l'interruption et la reprise sont résolues par l'adoption de ces conventions. Pour réaliser l'identification, on doit en plus établir une correspondance avec ces points particuliers de synchronisation et un identificateur d'unité logique de travail. Cet identificateur devra être transmis entre utilisateurs. Pour réaliser le prolongement d'une activité sur une connexion ultérieure il suffit, à l'ouverture de la connexion, de référencer le dialogue précédent ainsi qu'un numéro de synchronisation particulier (correspondant à la reprise). Pour réaliser un abandon, il suffit de se resynchroniser en précisant les unités de dialogue qui doivent être abandonnées. Notons qu'un mécanisme de jeton supplémentaire doit être instauré. En effet, seul l'utilisateur possédant le contrôle peut abandonner l'activité, alors que le mécanisme utilisé à cette fin (la resynchronisation) n'est soumise à aucun jeton.

Il serait donc possible, au niveau supérieur, de séparer le dialogue en unités logiques de travail identifiées et structurables, par l'intermédiaire de points de synchronisation majeurs. Cela impliquerait toutefois une gestion supplémentaire assurant l'identification des unités à la sémantique précise attachée à chaque point de synchronisation, c'est-à-dire une nouvelle série de conventions. Seule ombre au tableau, la resynchronisation (utilisée en tant que telle) remettrait en cause l'intégrité des "pseudo-unités logiques de travail" et de leurs "pseudo-unités de dialogue". De plus, dans cette perspective, si un utilisateur décide d'abandonner l'activité en cours, le fournisseur n'a aucun moyen de l'en empêcher.

Nous constatons ainsi que le concept d'unité logique de travail est un concept à part entière, qui permet aux utilisateurs du service session une structuration plus forte du dialogue, englobant la structuration de base (majeure et mineure). Cette structuration du dialogue, tout en restant générale, décharge les niveaux supérieurs de toute une série de conventions.

2.9.4 Exemples d'utilisation

On retrouve le concept complet d'activité, dans la norme T62, sous la forme de document télétex. L'envoi d'un document télétex constitue une activité. Une fois le document envoyé, le contrôle peut être donné au correspondant pour l'envoi d'un autre document.

Il existe une très grande similitude entre l'emploi du concept de document télétex dans T62, et l'emploi du concept d'activité dans X400. En effet, l'envoi d'un message constitue, lui aussi, une activité. De plus, le mécanisme proposé suppose que le contrôle de la transmission n'est cédé qu'en dehors d'une activité.

La figure III.12 est reprise de la norme X410. Rappelons que cette norme décrit, entre autres, l'utilisation des services session, faite par le RTS dans le cadre du MHS. Cette figure est le diagramme d'état des séquences permises des primitives du service session pour le RTS émettant. On évolue d'un état à un autre, suite à l'occurrence d'un événement. Dans notre cas, il s'agit soit d'une requête, soit d'une confirmation. Une fois que l'émetteur a demandé la primitive de démarrage d'activité, il peut envoyer son message. L'envoi du message comporte les émissions successives d'une page et d'un point de synchronisation mineur. La taille de la fenêtre pour ce schéma est de 3. Si après trois envois l'émetteur n'a toujours pas reçu de confirmation de mineur, il se trouve bloqué en état 8. Les confirmations arrivent en fait de façon asynchrone (états 4,5,6,7,8,9,10,11). Notons que dans cet exemple, tout mineur doit être explicitement confirmé. Lorsque l'émetteur en a terminé, il demande la terminaison de l'activité (état 3 ou 5 ou 7). Il peut encore recevoir une (état 10) ou deux (état 11) confirmations de mineur. A tout moment, l'émetteur peut détecter ou être averti d'une erreur. Dans ces cas, il interrompt ou abandonne l'activité (état 12, 13).

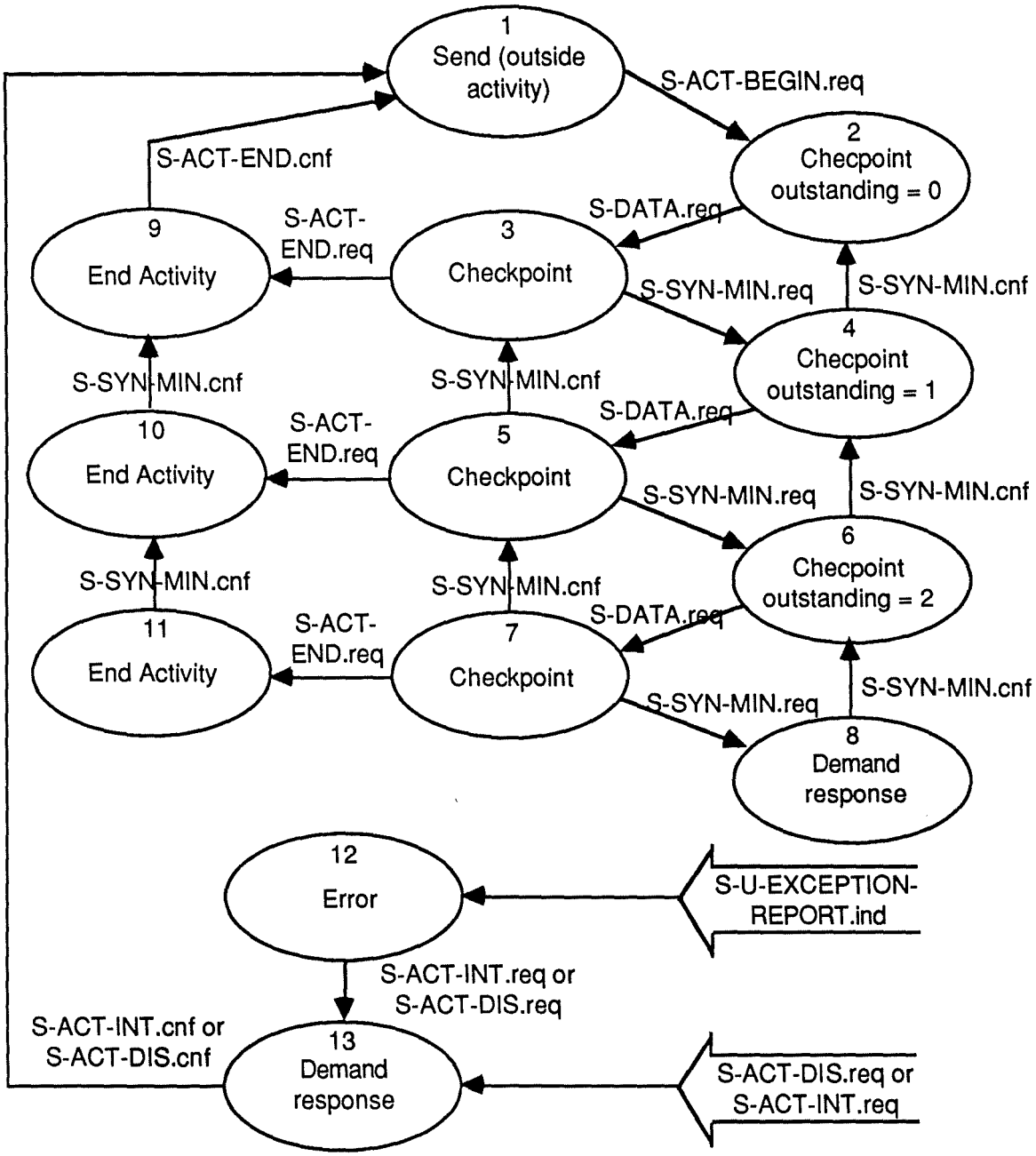


Fig III.12 : utilisation des services session par RTS (émetteur)

3 Traitement d'erreur

Nous allons analyser, dans ce troisième point, les services liés au traitement d'erreur, ainsi que la façon dont on doit ou on peut s'en servir.

Nous allons d'abord établir un classement de ces services d'après des critères de comparaison. Ensuite, nous classerons les erreurs. Nous leur appliquerons alors les mécanismes de gestion d'erreur. Finalement, nous apporterons quelques remarques importantes.

3.1 Mécanisme de gestion d'erreur

La norme ne nous est pas apparue très claire au niveau de la gestion d' erreur : les services concernés ainsi que leur utilité sont éparpillés dans le texte. Dans l'espoir d'y apporter plus de clarté, nous allons dans un premier temps répertorier ces services. Ensuite, nous les classerons d'après certains critères.

3.1.1 Répertoire des services

Dans le répertoire des services, on retrouve plusieurs types de service de gestion d'erreur. Certains services sont purement de signalisation : ils sont utilisés pour informer les utilisateurs de la survenance d'un problème qui doit être résolu par eux. Dans cette première catégorie, on retrouve les services suivants :

- signalisation d'anomalie par le fournisseur
- signalisation d'anomalie par l'utilisateur.

D'autres services sont purement de "remède" (restauration) : ils sont utilisés pour résoudre l'état d'anomalie dans lequel se trouve le fournisseur session. Dans cette deuxième catégorie, on retrouve les services qui permettent de céder le jeton de données.

Enfin, il existe des services qui tiennent à la fois de la signalisation et du remède : leur invocation résulte de la présence d'un problème et véhicule le moyen proposé pour le résoudre. Cette troisième catégorie comprend les services suivants :

- resynchronisation
- interruption et reprise d'activité
- abandon d'activité
- coupure par l'utilisateur ou le fournisseur.

3.1.2 Classification des services

Maintenant que nous avons répertorié les services, nous allons les classer selon deux critères : premièrement, la prépondérance entre deux services, deuxièmement, les conséquences de l'utilisation d'un service au niveau du dialogue.

3.1.2.1 Application du critère 1

Définissons d'abord une prépondérance entre deux services de gestion d'erreur. Entre un service A et un service B, A est dit prépondérant lorsque l'un des deux sous-critères suivants est satisfait :

- la demande de A bloque la demande de B, c'est à dire : tant que le service A n'est pas totalement réalisé, le service B ne peut être réalisé
- lorsqu'un conflit éclate (collision des demandes de service), A l'emporte sur B, c'est à dire : des deux services, c'est A qui est réalisé.

- Application du premier sous-critère

Les services de cession de jeton ne sont bloquants pour aucun service. Les services de signalisation d'anomalie bloquent l'ensemble des services sauf ceux de remède. Les services de resynchronisation bloquent l'ensemble des services sauf ceux de fin anormale d'activité et de connexion. Les services de fin anormale d'activité bloquent l'ensemble des services sauf celui de coupure. L'emploi du service de terminaison anormale de connexion (coupure) est toujours permis.

- Application du second sous-critère

La norme détermine la procédure à suivre lors de la collision de certaines demandes de service (tableau 29 du service, tableau 8 du protocole). Nous en avons déduit le classement qui suit. On y retrouve par ordre croissant de notre sous-critère :

- la resynchronisation option redémarrage
- la resynchronisation option choix de l'utilisateur
- la resynchronisation option abandon
- l'interruption d'activité
- l'abandon d'activité
- la coupure.

Notons que, sauf pour l'option redémarrage, lorsque deux demandes de resynchronisation de même type entrent en collision, l'appelant l'emporte sur l'appelé.

3.1.2.2 Application du critère 2

La signalisation d'anomalie est un moyen "doux" de signaler une erreur : le fournisseur est mis dans un état d'anomalie d'où il ne sortira que par un service de remède.

La cession de jeton de données est un remède "doux" : seul le jeton de données est cédé par un utilisateur; les autres variables de la SPM (autres jetons, numéro de série) restent inchangées.

La resynchronisation est un mécanisme plus conséquent. L'ensemble des variables est modifié. De plus, elle porte sur les unités et les sous-unités de dialogue. Notons qu'une négociation intervient entre les deux utilisateurs par le choix des nouvelles valeurs à affecter aux variables.

La terminaison anormale d'une activité est un remède plus brutal. Elle implique la prise de l'ensemble des jetons et porte sur l'unité logique de travail en cours. Il n'y a pas de négociation sur le numéro de série ni de refus possible.

Vient finalement le remède le plus radical, la coupure. Elle porte sur une connexion. De plus, il n'y a pas de refus possible.

3.1.2.3 Classification définitive

En regroupant les différents critères, on retrouve, par ordre croissant de prépondérance et de conséquence :

- la signalisation d'anomalie
- la cession du jeton de données
- la resynchronisation option redémarrage
- la resynchronisation option choix de l'utilisateur
- la resynchronisation option abandon
- l'interruption et la reprise d'activité
- l'abandon d'activité
- la coupure de la connexion.

Par exemple, cela revient à dire que, si une demande d'abandon d'activité est en cours, toute demande de resynchronisation sera ignorée, et qu'un abandon d'activité a plus de conséquences sur le dialogue qu'une resynchronisation.

3.2 Classification des erreurs

Une façon de recenser les types d'erreurs possibles est d'analyser les primitives de gestion d'erreur. La plupart de ces primitives comportent un paramètre raison, codant l'erreur détectée. Il s'agit des primitives de signalisation d'anomalie, de terminaison anormale d'activité et de coupure par le fournisseur du service session. Les autres primitives (de resynchronisation, de coupure par l'utilisateur et de cession de jeton) possèdent un paramètre données de l'utilisateur, dans lequel il est possible d'insérer le motif d'utilisation de la primitive. Pour ces dernières primitives, la norme en précise quelques raisons d'utilisation.

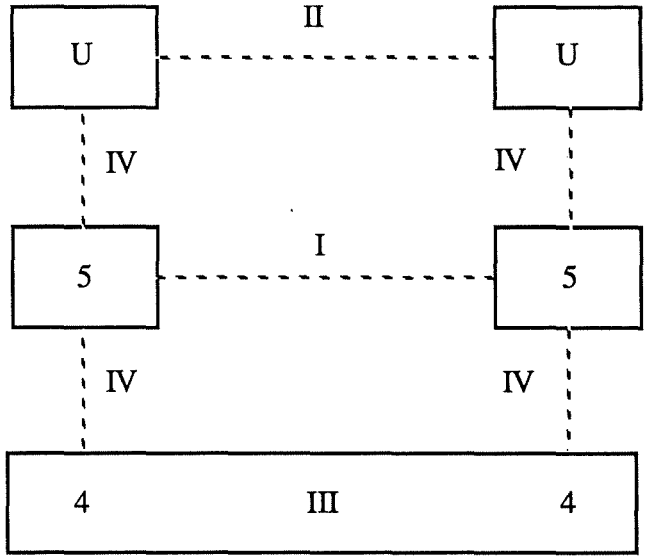
Voici, d'après la norme, les différentes erreurs "invocables" lors de l'utilisation des services de :

- signalisation fournisseur : erreur de protocole, erreur non-spécifique
- signalisation utilisateur : capacité de réception compromise, erreur locale de l'utilisateur, problème de séquence , demande du jeton de données, erreur de procédure irrémédiable, erreur non spécifique
- coupure fournisseur : déconnexion du transport , erreur de protocole, indéterminé
- coupure utilisateur : indéterminé
- interruption activité : capacité de réception compromise, erreur locale de l'utilisateur, problème de séquence , demande du jeton de données, erreur de procédure irrémédiable, erreur non spécifique
- abandon activité : capacité de réception compromise, erreur locale de l'utilisateur, problème de séquence , demande du jeton de données, erreur de procédure irrémédiable, erreur non spécifique
- resynchronisation : erreur , absence de réponse de l'utilisateur ou du fournisseur, désaccord entre utilisateurs.

A la première lecture, on constate que les raisons citées sont pour la plupart soit vagues, soit très semblables. En effet, comparons les liste d'erreurs "invocables" lors d'une signalisation d'anomalie, d'une interruption ou d'un abandon d'activité : ces listes

sont identiques. Reprenons aussi certaines erreurs de la liste : erreur, indéterminé, erreur non-spécifique, etc ...

Le texte de la norme relatant les raisons d'erreur ne nous ayant pas satisfait, nous avons dès lors tenté de les repérer en analysant systématiquement les différents endroits dont elles pouvaient provenir.



U : utilisateur ---- : dialogue I : type d'erreur

Fig III.13 : typologie des erreurs

Le schéma III.13 représente le niveau 5, les utilisateurs, le fournisseur transport, et les différents dialogues. Les chiffres romains situent les différents types d'erreur. Une erreur de type I résulte d'une erreur de procédure dans le protocole session ou de la réception d'un PDU non-valide. Une erreur de type II reflète un désaccord entre les utilisateurs au niveau de leur protocole (exemple : une entité 6 se sert d'un codage non-autorisé lors de la négociation), ou un état d'exception (exemple : un utilisateur doit interrompre le dialogue pour exécuter une tâche plus prioritaire). Une erreur de type III est soit une erreur irrécupérable détectée par le niveau transport et signalée à la couche session (exemple : la communication réseau a été coupée), soit une erreur non-détectée par la couche transport (exemple : perte d'un PDU non-détectée). Une erreur de type IV résulte d'une erreur locale dans l'utilisation d'un service (exemple : dans un paramètre, la taille des données excède la limite permise).

3.3 Traitement des erreurs

Les erreurs de type I et celles irrécupérables de type III sont réglées par la norme et traitées par le fournisseur session. Pour les autres erreurs de type III, tout dépend de l'endroit où on les détecte (si on les détecte). Pour les erreurs de type II, la norme offre des services de gestion d'erreur, l'initiative du traitement étant laissée aux utilisateurs. Les erreurs de type IV sont laissées à l'initiative locale.

3.3.1 Traitement des erreurs de type I

Les erreurs de type I peuvent être traitées par les deux services suivants :

- coupure par le fournisseur
- signalisation d'anomalie par le fournisseur.

Le premier mécanisme peut toujours être employé. Le second subit certaines restrictions (dues au contrôle du jeton de données, à la structuration en activités). Si le choix est possible entre ces deux mécanismes, il est laissé à l'initiative du fournisseur. Sinon, la connexion est coupée par le fournisseur.

Remarquons que le premier remède (coupure) est radical : on avertit les utilisateurs et on coupe la connexion. Le second est un mécanisme de signalisation, laissant la possibilité aux utilisateurs de "réparer" la connexion session.

Pour illustrer ce type d'erreur, mettons nous à la place d'un concepteur de SPM. Nous pouvons décider que, pour toute erreur de protocole (exemple : réception d'un PDU non-valide), la SPM coupera la connexion. Nous pouvons aussi décider que, si les circonstances le permettent (voir restrictions), pour toute erreur portant sur le numéro de série de synchronisation (exemple : rupture de séquence), la SPM se limite à informer l'utilisateur de la survenance d'une anomalie.

3.3.2 Traitement des erreurs de type II

On peut se retrouver face à une erreur de type II, soit :

- suite à un désaccord de niveau supérieur (suite à une erreur de protocole, à un autre événement tel qu'une perte de transmission détectée à ce niveau)
- parce que le désaccord a été signalé par le fournisseur (suite à ce désaccord, un utilisateur commet une erreur dans l'emploi des services de niveau session)
- suite à un événement "exceptionnel" (arrêt pour exécuter une tâche plus prioritaire).

Ces erreurs sont traitées par l'ensemble des services de gestion d'erreur. Nous avons vu que les conséquences de ces mécanismes variaient avec leur degré de prépondérance. On emploiera un de ces mécanismes, d'après leur disponibilité et la gravité de l'erreur (à problème grave, remède radical).

Illustrons le choix d'un service de traitement d'erreur par un exemple concret. Un utilisateur envoie un texte à un autre utilisateur. Supposons qu'ils se synchronisent par la pose de points mineurs, et qu'ils déterminent une taille de fenêtre égale à 3 (nombre de points mineurs pouvant être en attente de confirmation, avant que le transfert de données ne soit suspendu). Une fois l'échange commencé, l'émetteur, après avoir posé trois points mineurs, arme un "timer" pour éviter une attente infinie de confirmations. Son correspondant ne lui ayant pas répondu (peu importe la raison), le "timer" arrive à expiration. L'émetteur décide alors de resynchroniser, et arme de nouveau le "timer", limitant ainsi l'attente de la confirmation de resynchronisation. Son correspondant ne lui répondant toujours pas, suite à l'expiration du timer, il coupe la connexion.

3.3.3 Traitement des erreurs de type III

Parmi les erreurs de type III, on retrouve soit les erreurs irrécupérables détectées par le fournisseur transport (coupure réseau) et signalées à la session, soit les erreurs non-détectées (perte d'un PDU). Dans le premier cas, le mécanisme utilisé est la signification, par chaque SPM à son utilisateur respectif, d'une coupure par le fournisseur. Dans le deuxième cas, tout dépend de l'endroit où l'erreur est détectée : si c'est le niveau 5 qui la détecte, on revient à un traitement d'erreur de type I; si c'est l'utilisateur, on revient à un traitement d'erreur de type II. Remarquons aussi qu'une pareille erreur peut ne jamais être détectée.

3.3.4 Traitement des erreurs de type IV

Le traitement des erreurs de type IV est laissé à l'initiative locale. L'erreur résulte, en fait, de l'utilisation erronée d'une primitive de service, provenant :

- d'une erreur de programmation
- d'une tentative permise d'utilisation.

On peut envisager que, à la moindre occurrence de ce genre d'erreur, le fournisseur du service session coupe la connexion. Cette mesure draconienne est utilisable lors de la mise au point d'un programme (utilisant ou réalisant la couche session). Il parait cependant préférable de signaler l'erreur par un mécanisme local, ou par le service approprié. Le traitement de l'erreur est ainsi laissé à l'initiative de l'utilisateur.

Reprenons l'exemple des deux utilisateurs qui se transfèrent des textes. Supposons que le transfert se fait à l'alternat. Lorsqu'il n'y a pas d'échange en cours, les deux utilisateurs ont le droit d'essayer de transférer un texte. Hors, vu l'existence du jeton, un des deux utilisateurs se verra refuser sa demande par le fournisseur. Il est préférable de laisser cet utilisateur réagir plutôt que de prendre une décision au niveau de la session.

3.4 Remarques

Nous tenons à souligner la cohérence générale du traitement d'erreur. Les mécanismes de traitement d'erreur sont liés aux mécanismes de structuration du dialogue. Premièrement, la prépondérance de ces mécanismes varie en fonction du niveau de structure concerné. Deuxièmement, les conséquences du traitement d'erreur choisi sur le dialogue sont proportionnelles au niveau de structuration dont il fait l'objet.

Rappelons, par contre, les limitations imposées par les concepteurs de la norme à l'emploi de la signalisation d'anomalie. Ce mécanisme, pourtant bien utile aux deux utilisateurs, ne peut être utilisé que par une application ou une SPM à la fois, et encore, si l'unité fonctionnelle half duplex a été sélectionnée. Ceci implique que, pour toute erreur de protocole, la connexion sera coupée sans autre intervention possible des utilisateurs. De plus, l'utilisateur ayant la main devra utiliser un autre mécanisme pour signaler la survenance d'une erreur. A notre avis, ceci constitue une lacune.

Au contraire d'une autre norme comme T62 qui utilise plusieurs 'timer' tel que celui d'inactivité, la session ISO ne propose l'emploi que d'un seul 'timer', et ce, pour les mécanismes de coupure et de déconnexion. Pour toute demande bloquante et nécessitant une réponse, un système doit être mis en place, soit au niveau session, soit ailleurs, pour éviter une attente infinie.

Comme le fait remarquer Fausto Canenchi dans son article [CAN 86], l'emploi d'un 'timer' est nécessaire dans un cas bien précis de 'deadlock'. En effet, prenons le cas où l'appelant, ayant la main, envoie un SPDU de resynchronisation. Suite à un problème quelconque, l'appelé ne comprend pas ce SPDU et lui répond par un SPDU de signalisation d'anomalie. L'appelant, recevant ce SPDU, n'en tiendra pas compte. Les deux SPM sont donc en attente. "La solution d'introduire un 'timer' pour résoudre ce 'deadlock', a été jugée comme non satisfaisante par le groupe session ISO". [CAN 86]

Hormis la limitation imposée au mécanisme de signalisation d'anomalie et la non-utilisation d'un 'timer' d'inactivité, la gestion d'erreur est assez riche. Ceci sous-entend que, aussi bien au niveau de la SPM, mais surtout au niveau application, une analyse très fine peut être réalisée sur les causes d'une erreur et sur le mécanisme adéquat à employer, ce qui peut entraîner un supplément important de programmation.

Remarquons enfin que, à une exception près, l'ensemble des services proposés mettent la SPM en état d'anomalie. Ceci signifie que l'utilisateur n'a le droit que de "réparer" cet état : l'échange est bloqué par le fournisseur tant que l'erreur n'est pas traitée.

4 Analyse des sous-ensembles types

Au premier chapitre, nous avons introduit les trois sous-ensembles types de service session proposés par la norme, à savoir : BCS (Basic Combined Subset), BSS (Basic Synchronized Subset) et BAS (Basic Activity Subset). Nous allons maintenant les analyser. Nous allons essayer de déterminer à quelle genre d'applications ils sont destinés et d'en expliquer les raisons.

Pour ce faire, nous rappellerons la composition de ces sous-ensembles. Nous les examinerons un à un, dans l'ordre donné ci-dessus. Ensuite, nous analyserons les possibilités de choix laissées à deux grandes catégories d'applications : celles qui désirent un contrôle half duplex (applications half duplex) et celles qui ne le nécessitent pas (applications full duplex). Nous terminerons par quelques remarques.

4.1 Etude de la composition des sous-ensembles types

Le tableau III.3 rappelle la composition des sous-ensembles types. Il permet de mettre en évidence les différences qui existent entre ces différents regroupements d'unités fonctionnelles.

Unités fonctionnelles	BCS	BSS	BAS
Noyau	X	X	X
Full duplex	X	-	-
Half duplex	X	X	X
Signalisation d'anomalie	-	-	X
Terminaison négociée	-	X	-
Données typées	-	X	X
Données expresses	-	-	-
Synchronisation mineure	-	X	X
Synchronisation majeure	-	X	-
Resynchronisation	-	X	-
Gestion d'activité	-	-	X
Information de capacité	-	-	X

Tableau III.3 : composition des sous-ensembles

4.1.1 BCS

Une des caractéristiques fondamentales de BCS est l'absence de structuration du dialogue. Notons aussi qu'il est le seul à proposer les unités half et full duplex. Une autre caractéristique importante est qu'il permet dans son mode half duplex, de contrôler réellement le caractère alternatif du dialogue. En effet, en ce qui concerne la phase de transfert, il ne permet que l'envoi de données normales, la passation du jeton et sa demande. Ces trois services sont contrôlés par le fournisseur grâce au jeton de données. Le troisième permet aussi de transférer des données, mais ce n'est pas là son but principal. Il est supposé que les utilisateurs emploient ce troisième service pour son but avoué. Notons aussi que le BCS n'inclut pas la terminaison négociée. Dans ce cas précis, si un utilisateur décide d'arrêter la communication, l'autre ne peut refuser.

Le traitement d'erreur proposé dans BCS est tout à fait élémentaire : le seul mécanisme disponible est la coupure de la connexion. A défaut d'un système local ou transparent au fournisseur, suite à toute erreur, la communication sera donc coupée. Remarquons toutefois que les procédures à respecter par une SPM dans le cadre d'un échange en BCS sont très simples. Vu cette remarque, il est raisonnable de couper la connexion suite à une erreur de protocole de niveau 5 (type I).

Un utilisateur choisira donc BCS s'il désire que le fournisseur n'exerce aucun contrôle sur sa transmission, ou bien, seulement sur le caractère alternatif du dialogue.

4.1.2 BSS

BSS permet de structurer l'échange en unités et sous-unités de dialogue. Une caractéristique importante est qu'il n'inclut que l'unité half duplex (contrôle du jeton de données obligatoire). Notons que ce sous-ensemble permet d'envoyer des données de deux façons conventionnelles (grâce à des services spécialement dédiés à cette effet) : par le service d'envoi de données normales qui subit le contrôle du jeton, et par le service de données typées qui transgresse ce contrôle. Notons enfin qu'il inclut l'unité fonctionnelle de terminaison négociée.

Ce sous-ensemble propose la resynchronisation comme mécanisme principal de traitement d'erreur. Il n'inclut pas la signalisation d'anomalie. Nous en connaissons les conséquences au niveau du protocole.

Un utilisateur choisira donc BSS si le dialogue envisagé sur une connexion est composé d'une seule unité logique de travail fortement structurée. Il permet à son correspondant de lui transmettre des données même quand ce dernier n'a pas la main. Il peut se réserver le droit de demander la terminaison normale de la connexion et permet à son correspondant de la lui refuser de façon conventionnelle ou par resynchronisation. Il doit accepter la non-sélection de la signalisation d'anomalie.

4.1.3 BAS

La caractéristique principale de BAS est qu'il comprend l'unité fonctionnelle de gestion d'activité. Il permet de structurer les unités logiques de travail grâce à la synchronisation mineure. Une autre caractéristique importante est qu'il n'inclut que l'unité fonctionnelle half duplex (contrôle du jeton de données). Notons qu'il permet d'envoyer des données de trois façons conventionnelles par les services d'envoi de données normales, typées et d'information de capacité. Remarquons qu'il n'inclut pas la terminaison négociée.

Ce sous-ensemble propose comme traitement d'erreur les mécanismes liés à l'activité (abandon, interruption et reprise). Il ne comprend pas la resynchronisation. Ceci implique qu'il est impossible de reprendre le dialogue en un point de synchronisation mineur tout en restant dans l'activité courante; il faut pour cela interrompre et reprendre l'unité logique de travail. Remarquons enfin que la signalisation d'anomalie a été retenue. Nous en connaissons les avantages.

Un utilisateur choisira donc BAS si le dialogue envisagé dans le cadre d'une connexion est composé de plusieurs unités logiques de travail, composée chacune d'une seule unité de dialogue structurable. Il permet à son correspondant de lui transmettre des données même quand ce dernier n'a pas la main (utilité : voir III.2.2.1). Il peut se réserver le droit demander la terminaison normale de la connexion mais ne permet en aucune façon à son correspondant de la lui refuser. Enfin, il doit accepter les conséquences du traitement d'erreur citées plus haut.

4.2 Applications half et full duplex

Examinons les possibilités offertes par ces trois sous-ensembles aux applications qui désirent le contrôle half duplex (applications half duplex) ou qui ne le nécessitent pas (applications full duplex).

Pour une application half duplex, le choix est complet : les trois sous-ensembles permettent ce type de dialogue. Il dépendra alors d'autres caractéristiques telles que la structuration, le niveau de contrôle half duplex (seul BCS contrôle réellement le caractère alternatif du dialogue), les performances attendues.

Pour une application full duplex, le choix est restreint : en principe, il est limité à BCS. Notons toutefois que les deux autres sous-ensembles proposent l'utilisation des services de données typées dont le caractère est full duplex. Il est donc possible pour une application ne nécessitant pas de contrôle sur l'envoi de données d'opter pour n'importe quel sous-ensemble type.

4.3 Remarques

La composition des trois sous-ensembles a été pour nous la source de nombreuses questions dont certaines restent encore sans réponse. Pourquoi BSS et BAS comprennent-ils uniquement l'unité fonctionnelle half duplex? Nous pensons que c'est dû au fait que, dans les deux cas, la structuration proposée ne porte que sur un seul flux de données à la fois. Pourquoi avec BAS ne peut-on pas refuser la terminaison de la connexion? Pourquoi ce dernier sous-ensemble n'inclut-il pas la resynchronisation? Nous croyons que cela provient de l'intégration de la norme T62. En effet, dans ce protocole, il n'est pas possible de refuser la terminaison de la connexion. De plus, il n'y a pas de correspondance entre la resynchronisation ISO et les mécanismes proposés dans T62. Pourquoi BSS inclut-il alors l'unité fonctionnelle de terminaison négociée? Pourquoi se voit-il privé du mécanisme de gestion d'erreur bien utile qu'est la signalisation d'anomalie? Nous n'avons pas trouvé de réponse à ces deux dernières questions.

Dans le premier chapitre, nous avons insisté sur le caractère "Basic" des sous-ensembles BCS, BSS et BAS. Citons maintenant trois catégories d'application qui ne sont pas visées par ces regroupements :

- un dialogue full duplex structuré dans les deux sens
- un échange composé de plusieurs activités comprenant plusieurs unités de dialogue structurables ou non
- un échange composé d'une seule unité de dialogue structurable.

Ainsi que nous l'avons constaté dans certaines revues, quelques personnes confondent les notions noyau et BCS. Nous nous permettons dès lors de souligner la différence entre ces deux notions. Le noyau est l'unité fonctionnelle qui regroupe les services minimaux session. BCS est le sous-ensemble (combinaison valide d'unités fonctionnelles) minimal proposable par une application half ou full duplex.

5 Choix d'un sous-ensemble

Nous allons maintenant aborder la problématique liée au choix, par un utilisateur, d'un sous-ensemble particulier convenant aux besoins de son application. Le but est ici d'aider les futurs utilisateurs dans leur sélection tout en soulignant la difficulté de leur tâche.

Nous allons pour cela introduire cette problématique et faire apparaître des critères de choix. Nous proposerons alors une démarche à suivre. Afin d'illustrer l'ensemble, nous appliquerons la méthode proposée à des "caricatures" d'applications téléinformatiques .

5.1 Problématique et critères

Nous avons vu que le fournisseur session proposait toute une série d'outils standardisés sur lesquels il exerce un contrôle. L'utilisateur doit décider s'il désire utiliser ces outils, respecter les procédures d'utilisation et le contrôle qui y sont attachés. S'il ne le désire pas, il s'oriente alors vers le sous-ensemble BCS (III.4.1.1). Dans le cas contraire, se présentent à lui quatre grands types de choix : le type de structuration, la façon d'envoyer des données, le traitement d'erreur et la manière de terminer la connexion. Il choisira dans ces quatre groupes les services qui, d'après leurs caractéristiques, se rapprochent le plus du dialogue qu'il envisage. Ces caractéristiques ont été analysées précédemment.

Le problème est alors le suivant : le choix d'un service peut avoir des conséquences sur les caractéristiques et l'utilisation d'un autre service. En effet, nous avons vu que le traitement d'erreur est fonction du niveau de structuration, que l'adoption de la resynchronisation a des conséquences sur la procédure de terminaison de la connexion, que les jetons de structuration, de données et terminaison sont liés ... L'ensemble de ces conséquences devra être pris en compte par l'utilisateur.

5.2 Démarche proposée

Afin de résoudre le problème du choix, nous proposons la démarche suivante :

- analyse des caractéristiques de structure de la communication à établir dans le cadre de l'application et établissement d'une correspondance entre cette structure et la structuration session
- analyse du rapport entre utilisateurs
- établissement des besoins réels en services session de structuration, d'envoi de données, en contrôle half duplex, en traitement d'erreur ainsi qu'en ce qui concerne la procédure de déconnexion, tout en vérifiant les conséquences de la combinaison d'unités fonctionnelles choisies.

5.3 Application de la démarche

On parle "beaucoup" dans la littérature de quatre applications téléinformatiques standard : FTAM (File Transfer Access and Management), X400 (messagerie électronique), JTM (Job Transfer Management) et VT (Virtual Terminal).

Notre but n'est pas d'étudier en détail ces applications. Nous allons seulement en définir certaines caractéristiques et émettre des hypothèses les concernant. Nous appliquerons ensuite à ces quatre familles d'applications notre méthode d'analyse.

5.3.1 Pseudo FTAM : accès et transfert de fichiers

- Introduction

"FTAM est un standard de l'ISO défini pour transférer, accéder et gérer l'information enregistrée ou échangée entre systèmes, comme fichier". [LEW 83]

Les services de FTAM permettent, entre autres, à un utilisateur d'accéder à un fichier (un seul à la fois) pour réaliser des opérations sur l'entièreté du fichier tel que créer un fichier, changer les attributs d'un fichier (exemple : nom du fichier), ou des opérations sur le contenu du fichier (exemple : lire un élément du fichier ou l'ensemble du fichier).

Afin de simplifier et de clarifier l'application de notre méthode, notre étude ne portera que sur une caricature de FTAM : un pseudo FTAM.

- Spécification

Notre pseudo FTAM doit permettre à un utilisateur d'accéder à un serveur de fichiers éloigné. L'utilisateur ne peut accéder qu'à un seul fichier à la fois. Une fois qu'il a accédé à ce fichier, il doit pouvoir manipuler les attributs du fichier (modification d'un attribut), manipuler (lecture,écriture) soit un élément, soit l'ensemble du fichier.

- Etude des notions

- Accès à un fichier à la fois pour le manipuler.

On peut accéder à plusieurs fichiers consécutivement sur une connexion. Les actions commises lors de ces accès sont complètement séparées. Un fichier est identifié. On peut regrouper logiquement les actions réalisées sur un fichier. On peut éventuellement interrompre et reprendre l'ensemble des actions.

Conclusion : l'ensemble des manipulations d'un fichier peut être considéré comme une unité logique de travail identifiée, cad une activité session.

- Manipulations exercées sur les éléments et les attributs d'un fichier.

Toute manipulation ne peut être effectuée que lorsqu'un fichier a été sélectionné. Une manipulation peut être considérée comme un tout et est séparée d'une autre manipulation. On exerce une manipulation à la fois sur un fichier. Plusieurs actions consécutives peuvent faire partie d'une activité. Une manipulation peut être abandonnée.

Conclusion : l'échange d'information dans le cadre d'une manipulation peut être considéré comme une unité ou une sous-unité de dialogue, appartenant à une activité session.

- Manipulation d'un fichier.

Une manipulation de fichier (lecture d'un fichier) peut être considérée comme une seule manipulation, ou comme une suite de manipulations (lecture d'un élément d'un fichier). De toute façon, vu la taille importante que peut prendre un fichier, il est intéressant de pouvoir interrompre un transfert et de le reprendre par la suite, dans une même activité session ou dans une autre.

Conclusion : l'échange d'information dans le cadre d'une manipulation de fichier peut correspondre soit à une suite, soit à une seule unité ou sous-unité de dialogue

- Caractère maître-esclave du dialogue.

Une application (le demandeur) consulte un gestionnaire de fichiers (le serveur) auquel il soumettra des demandes. Une fois ses manipulations terminées, le demandeur clôturera la connexion. Nous sommes en présence d'un rapport de type maître-esclave : le demandeur ouvre la connexion, commence ou recommence une activité, exerce des manipulations, termine l'activité et, lorsqu'il en aura terminé avec le serveur de fichiers, clôturera la connexion.

L'envoi de données, résultant des demandes de manipulation du maître, se font dans un sens ou dans un autre (lecture ou écriture). Il fait partie d'une manipulation. Vu qu'on exerce qu'une seule manipulation à la fois, l'envoi de données résultant est purement de nature half duplex. Vu qu'il est encadré dans une manipulation, le transfert de données ne nécessite pas de contrôle supplémentaire half duplex au niveau session.

- Conclusions.

La structuration du dialogue correspond particulièrement bien à la découpe proposée par la session en unités logiques de travail ou activités. De plus, cette structuration est de nature half duplex, et donc, respecte le rapport maître-esclave des applications utilisatrices. Nous suggérons donc d'utiliser l'unité fonctionnelle de gestion d'activité.

Ainsi que nous l'avons vu, l'envoi de données ne nécessite pas de contrôle supplémentaire de jeton. Nous suggérons donc l'utilisation de l'unité fonctionnelle full duplex.

Afin de respecter le rapport maître-esclave existant aussi lors de la terminaison de la connexion, nous suggérons de ne pas utiliser l'unité fonctionnelle de terminaison négociée.

Nous avons vu que le contenu d'une activité était structuré en unités ou sous-unités de dialogue. Nous suggérons donc d'employer soit l'unité fonctionnelle de synchronisation majeure, soit celle de synchronisation mineure. Rappelons que ces structurations sont de nature half duplex, et que le droit de gérer l'activité est lié au droit de la structurer. Ceci respecte le rapport existant entre les deux utilisateurs session : un utilisateur, le maître, ouvre une activité afin de donner des ordres de manipulation de fichier.

- Choix d'un sous-ensemble session

- Choix entre synchronisation mineure et majeure

Si l'on considère qu'une manipulation correspond à une unité de dialogue, cela implique deux conséquences fâcheuses. Premièrement, il n'est pas possible de regrouper logiquement plusieurs manipulations par une structuration session. Deuxièmement, d'un point de vue plus pratique, à chaque fois qu'une opération aura été réalisée, une synchronisation majeure devra être effectuée, ce qui nous donne pour l'appelant :

- demande d'action
- réponse facultative de l'appelé
- synchronisation majeure
- réception de la confirmation.

Considérons maintenant qu'une manipulation est en fait une sous-unité de dialogue (entourée de deux points de synchronisation mineurs). Notons premièrement, que cette considération ne remet nullement en cause les caractéristiques d'une manipulation : la seule différence avec la première considération provient de la séparation moins nette entre manipulations. Ensuite, remarquons que les deux inconvénients cités ci-dessus disparaissent. En effet, le schéma d'une manipulation serait le suivant :

- pose de point de synchronisation mineur (bloquant ou non , au niveau supérieur)
- demande d'action
- réponse facultative de l'appelé à la demande d'action
- dans tous les cas, réception de la confirmation mineur.

Ceci permet aussi de regrouper plusieurs manipulations en unité de dialogue que l'on peut abandonner, dans laquelle on peut redémarrer. Remarquons que les primitives de synchronisation peuvent contenir la demande d'action. De plus, dans le cas ou la synchronisation serait non-bloquante, le maître peut émettre d'autres demandes d'action, indépendamment des réponses et des confirmations du serveur.

C'est pour l'ensemble de ces raisons que nous sélectionons l'unité fonctionnelle de synchronisation mineure

Qu'en est-il du problème de manipulation de fichier ?

Si on considère une manipulation de fichier comme une suite de manipulations d'éléments de fichier, notre solution convient particulièrement : ces manipulations sont regroupables en une unité de dialogue clairement séparée des autres, que l'on peut structurer, et dans laquelle on peut se resynchroniser.

Si on considère une manipulation de fichier comme une seule manipulation, une solution revient à considérer cette manipulation comme une méta-manipulation (entourée de points majeurs), respectant un des deux schémas suivants (côté maître) :

- schéma 1 (lecture) :
    - demande d'action
    - passation du jeton mineur
    - itération
        réception d'un point mineur (bloquant ou non, pour l'appelé)
        réception de données
        confirmation du point mineur
    - réception jeton mineur
    - synchronisation majeure

- schéma 2 (écriture) :
    - demande d'action
    - itération
        envoi d'un point mineur
        envoi de données
        réception de la confirmation du point mineur
    - synchronisation majeure.

Remarquons ici aussi, que les données peuvent être envoyées dans le SSDU de pose de point mineur, et que le maître peut continuer d'envoyer des données, indépendamment de la confirmation de l'esclave.

Nous en concluons que notre considération d'une manipulation, respecte le concept de manipulation, est plus performante, et ne limite en rien l'adoption d'une des deux hypothèses concernant la manipulation d'un fichier.

- Conséquence sur le traitement d'erreur session.

Si le choix de l'unité full duplex a des conséquences intéressantes au niveau de la structuration, il a des conséquences plutôt fâcheuses au niveau du traitement d'erreur du fournisseur session, puisqu'il empêche l'utilisation de l'unité fonctionnelle de signalisation d'anomalie. En effet, rappelons que la moindre erreur de protocole

entraînera la coupure de la connexion session. De plus, afin de signaler à l'autre utilisateur la détection d'un problème, une autre convention doit être adoptée :

- utilisation de la resynchronisation
- emploi d'un mécanisme tel que les données typées
- acceptation de cette conséquence.

Afin de permettre la reprise dans une activité, d'une manipulation, nous suggérons l'emploi de la resynchronisation. Notons aussi, à titre de détail, que la resynchronisation peut permettre à l'esclave de prendre le contrôle du dialogue. Lors de la réception de pareil indication, une resynchronisation de même type devra être demandée par le maître.

Remarquons enfin que, de par les choix précédents en unités fonctionnelles, nous disposons d'un mécanisme trés complet de traitement d'erreur.

- Sélection optionnelle des autres mécanismes session.

Trois unités fonctionnelles peuvent être encore sélectionnées :

- l'unité fonctionnnelle de données typées
- l'unité fonctionnnelle de données expresses
- l'unité fonctionnnelle de données de capacité.

Ces trois unités fonctionnelles peuvent être sélectionnées pour leurs caractéristiques propres.

- Conclusion

Notre proposition de sous-ensemble est donc la suivante :

- le noyau
- le full duplex
- la gestion d'activité
- la synchronisation mineure
- la resynchronisation

La norme FTAM ne propose pas de sous-ensemble précis de services de niveau session. Elle cite seulement ses besoins de la façon suivante : insertion de points de synchronisation, services de resynchronisation pour supporter le "file checkpointing" et la reprise en cas d'erreur. Nous ne pouvons dès lors établir de réelle comparaison entre notre combinaison d'unités fonctionnelles et les besoins en services de FTAM.

5.3.2 Pseudo X400 : messagerie de base

"Les systèmes de 'message-handling' fournissent des services de messagerie électronique en 'store-and-forward' dans le sens le plus général du terme. Ils permettent aux utilisateurs (soit une personne, soit un processus d'application) d'envoyer et de recevoir des messages. Un message consiste en une enveloppe portant des informations de contrôle de distribution et un contenu de message. Le contenu d'un message peut être un texte, un facsimile, des graphiques, de la voix, des structures arbitraires de données binaires, ou toute combinaison de ces types." [CUN 83]

Toujours afin de clarifier et simplifier l'application de notre méthode, notre étude ne portera que sur une messagerie de base dont la spécification évoluera au cours de notre analyse.

5.3.2.1 Spécification 1

Permettre l'envoi de messages, dans les deux sens de transfert, mais sur un seul sens et un seul à la fois, entre deux utilisateurs.

- Etude des notions

- Notion d'envoi de messages.

Un message est une suite identifiée de bits, plus ou moins importante, qui est destinée à être envoyée. Un message est un tout, clairement séparé d'un autre message. Vu la taille importante que peut prendre un message, il est intéressant de pouvoir poser des points de repère dans un message afin de se resynchroniser sur un de ces points en cas d'erreur.

Un seul message peut être envoyé à la fois, et ceci, sur un seul sens de transfert à la fois. On peut envoyer plusieurs messages en suivant. Vu la taille importante que peut prendre un message, son envoi peut être interrompu et repris par la suite. Le droit d'envoyer un message peut changer lors d'une connexion.

Conclusion : l'envoi d'un message peut être considéré comme une activité.

- Relations entre utilisateurs.

A l'ouverture de la connexion, le rôle des deux utilisateurs est équivalent : ils sont tous deux prêts à envoyer et à recevoir des messages. La seule différence entre eux résulte du fait que l'un est l'appelant et l'autre, l'appelé.

Pendant la phase de transfert, leur rôle va changer dynamiquement : ils deviendront tour à tour maître et esclave. En dehors de l'envoi d'un message, le contrôle peut être demandé et passé de l'un à l'autre.

Il se peut aussi que, à un moment donné, un utilisateur, ayant terminé l'envoi de tous ses messages, désire la fin de la connexion. Il doit pouvoir le signaler à l'autre. On peut considérer que seul l'appelant a le droit de terminer la connexion, ou bien que les deux utilisateurs, étant sur le même pied, ont tous deux le droit de demander la fin du dialogue.
On peut aussi considérer que, soit seul l'appelant ou l'appelé, soit les deux utilisateurs, ont le droit de refuser de terminer la connexion.

- Conclusions.

L'unité fonctionnelle de gestion d'activité convient particulièrement bien à la structure de l'échange et aux rapports existant entre les deux utilisateurs pendant la phase de transfert.

De plus, de par cette considération, le transfert de données, résultant de l'ouverture d'une activité, sera de nature half duplex. Ceci implique qu'aucun contrôle half duplex supplémentaire de niveau session n'est nécessaire.

Hélas, en ce qui concerne la terminaison, le modèle ISO ne laisse pas un éventail de possibilités aussi large. Afin de se rapprocher au plus près du rapport existant entre les deux utilisateurs, nous adopterons la terminaison négociée : l'utilisateur ne possèdant pas le contrôle peut signaler son intention de terminer la connexion par la demande du jeton de terminaison; l'appelant peut se différencier de l'appelé en ne cèdant jamais le jeton de terminaison.

Nous avons vu qu'il était intéressant de structurer un message afin de pouvoir reprendre le transfert en un point précis. Rappelons que deux mécanismes de structuration sont à notre disposition : la synchronisation mineure et majeure. Il faudra donc choisir entre ces deux services.

Notons que c'est celui qui détient le contrôle du dialogue, qui transfère les données. Il n'y a donc pas de problème d'interactions de jetons.

- Choix d'un sous-ensemble session

Rappelons qu'aucun contrôle supplémentaire de type half duplex de niveau session n'est nécessaire pour le transfert de données. Nous suggérons donc l'emploi de l'unité fonctionnelle full duplex.

Afin de respecter le rapport existant entre les utilisateurs, nous sélectionnons l'unité fonctionnelle de terminaison négociée.

- Choix d'une structuration.

Nous avons vu que la notion d'activité convenait particulièrement à l'envoi de message. Nous suggérons donc l'emploi de l'unité fonctionnelle de gestion d'activité.

Nous suggérons aussi l'emploi de la synchronisation mineure pour ses avantages au niveau des performances de transfert.

- Conséquences sur le traitement d'erreur session

L'unité fonctionnelle de gestion d'activité permet d'interrompre, de reprendre, ou d'abandonner une activité, en cas d'erreur ou de problème.

La resynchronisation permet de reprendre une unité de dialogue avec diverses options. Elle peut aussi permettre à l'esclave de signaler un problème lors du transfert du message. Le maître prendra alors les mesures qui s'imposent.

Le choix de l'unité full duplex a des conséquences importantes sur la gestion des erreurs : l'unité fonctionnelle de signalisation d'anomalie ne peut être sélectionnée. De

plus, afin de signaler une anomalie lors du transfert d'un message, l'utilisateur devra recourir à un autre mécanisme. Pour ces raisons, on aurait pu sélectionner l'unité fonctonnelle half duplex, au lieu de l'unité full duplex.

- Sélection optionnelle des autres mécanismes session.

Trois unités fonctionnelles peuvent encore être sélectionnées :

- l'unité fonctionnelle de données typées
- l'unité fonctionnelle de données expresses
- l'unité fonctionnelle de données de capacité.

L'unité fonctionnelle de données de capacité pourrait être employée pour tester les capacités de réception des deux utilisateurs entre chaque message.

Les deux autres unités fonctionnelles peuvent être employées de façon optionnelle, d'après leurs caratéristiques propres.

- Conclusion

Notre proposition de sous-ensemble est donc la suivante :

- le noyau
- le full duplex
- la gestion d'activité
- la synchronisation mineure
- la resynchronisation
- la terminaison négociée.

Contrairement à FTAM, la norme X400 précise les services de niveau session ainsi que la façon de les employer. Dans sa sélection, on retrouve les unités fonctionnelles suivantes :

- le noyau
- le half duplex
- la gestion d'activité
- la synchronisation mineure
- la signalisation d'anomalie.

Ce sous-ensemble ressemble étrangement à BAS. Ceci provient du fait que "les services session sélectionnés sont ceux qui fournissent une compatibilité avec T62" [CUN 83] dont nous avons déjà expliqué l'apparentement avec BAS.

Ces deux sous-ensembles comprennent la même structuration. Le traitement d'erreur est assez différent, ainsi que la terminaison. Notons aussi que X400 désire un contrôle half duplex de niveau session. C'est une question de choix. En ce qui concerne le traitement d'erreur, nous trouvons plus intéressant de reprendre le transfert dans une activité plutôt que de devoir l'interrompre et la reprendre en un point. Vu la sélection de l'unité fonctionnelle de resynchronisation, l'esclave dispose alors d'un moyen de signaler une anomalie. Quant au half duplex, il ne nous est pas paru nécessaire. Par contre, les concepteurs de T62 préfèrent eux jouïr de ce contrôle. Pour la terminaison, nous trouvons que la procédure de la terminaison négociée respecte plus le rapport entre utilisateurs.

5.3.2.2 Spécification 2

Permettre l'envoi de messages, dans les deux sens de transfert, un à la fois par sens de transfert, entre deux utilisateurs.

- Etude des notions

La notion de message est inchangée.

Par contre, la notion d'envoi de message perd une de ces caractéristiques fondamentales. En effet, il est maintenant possible d'envoyer un message et d'en recevoir un autre, en même temps.

Il n'existe hélas pas de mécanisme de structuration de niveau session permettant de rendre cette nouvelle notion. En effet, les mécanismes de structuration en activités et sous-unités de dialogue, ne peuvent s'exercer que sur un seul sens du transfert à la fois : le mécanisme de synchronisation symétrique ne permet que la découpe en unités de dialogue.

- Conclusions

Il en résulte que, si une application désire échanger des messages avec une autre application en même temps sur les deux sens de transfert, trois solutions s'offrent à elle.

Premièrement, elle peut réaliser un mécanisme de structuration étendu aux deux sens de transfert au niveau application. Ce mécanisme alourdira considérablement la réalisation de ce niveau et de plus, étant propre à cette application, il ne pourra être réutilisé par une autre.

Deuxièmement, on peut malgré tout utiliser la synchronisation symétrique. Il sera alors nécessaire de construire un mécanisme permettant d'identifier les diverses activités, de les structurer, de permettre leur reprise, leur interruption, leur abandon. Ce mécanisme aurait les mêmes inconvénients que celui de la première solution. En outre, ainsi que nous l'avons expliqué dans ce chapitre lors de la comparaison des

différents mécanismes de structuration, il n'est pas aisé d'utiliser un mécanisme de structuration pour un autre et de plus, cela peut avoir des conséquences fâcheuses au niveau des performances du transfert.

Troisièmement, on pourrait ramener cette spécification à une troisième spécification : permettre l'envoi de messages, un à la fois, sur un seul sens de transfert, entre deux utilisateurs. Cette dernière spécification, si elle ne change pas la notion d'envoi de message (hormis l'interdiction de changer le contrôle en dehors d'une activité), modifie le rapport existant entre les deux utilisateurs : l'appelant reste maître de l'ensemble du dialogue. On peut donc garder le sous-ensemble déduit de la première analyse, en y soustrayant l'unité fonctionnelle de terminaison négociée. Au niveau de l'utilisateur, on devra créer deux applications : la première s'occupant de la réception d'un message, la seconde permettant d'en envoyer.

5.3.3 Pseudo JTM : exécution de job à distance

L'OSI a créé un modèle de Job Transfer and Manipulation (JTM) afin de faciliter l'exécution de job dans des systèmes distribués. [LAN 83] JTM permet entre autres, à un utilisateur, de transférer un job et d'en demander son exécution, de demander de l'information sur l'exécution de ce job, de suspendre ou de reprendre son exécution, ou enfin de l'arrêter.

Toujours dans le souci de simplifier et de clarifier l'application de notre méthode, notre étude portera sur un JTM de base (pseudo JTM) dont la spécification évoluera au cours de notre analyse.

5.3.3.1 Spécification 1

Permettre à un utilisateur de transférer et de demander l'exécution d'un job (un seul à la fois), de demander de l'information sur l'exécution d'un job, de suspendre et de reprendre l'exécution d'un job, d'arrêter un job.

- Etude des notions

- Notion de job.

Un job comprend une suite identifiée de bits (programme, données, information de contrôle), plus ou moins importante, qui est destinée à être transférée et exécutée. Vu la taille importante que peut prendre un job, il est intéressant de pouvoir poser des points de repère lors du transfert du job, afin de pouvoir reprendre le transfert par la suite.

Suite au transfert du job et à sa demande d'exécution, plusieurs demandes d'action sont possibles, par exemple : demande d'information sur l'exécution d'un job. Les résultats de ces actions doivent être transférés.

A la fin de l'exécution du job, les résultats de l'exécution devront être transférés. La taille de ces résultats peut être importante.

- Exécution éloignée d'un job.

L'exécution d'un job par un correspondant comprend le transfert du job et sa demande d'exécution, des demandes optionnelles d'information (sur l'état d'avancement de l'exécution) impliquant des réponses, l'envoi du message de terminaison et des résultats du job. Elle constitue un tout décomposé en plusieurs unités. Elle peut être interrompue et reprise par la suite. Plusieurs exécutions peuvent être demandées consécutivement, une seule à la fois.

Conclusion : l'exécution d'un job par un correspondant peut être considérée comme une unité logique de travail, comprenant plusieurs unités de dialogue ou sous-unités de dialogue.

- Rapport entre utilisateurs.

Le rapport entre utilisateurs est de type maître-esclave pendant toute la connexion. Le maître ouvre une connexion afin de soumettre une ou plusieurs demandes d'exécution de job et attend les résultats. Lorsqu'il en aura terminé, il clôturera la connexion.

Le transfert d'information est conséquent à une demande du maître. Au départ, le maître transfère un fichier contenant le job. Par la suite il reçoit les résultats d'une demande (information sur le job, résultats du job). Le transfert de données est donc de nature half duplex et ne nécessite pas de contrôle supplémentaire de niveau session.

- Choix d'un sous-ensemble session

- Choix des unités fonctionnelles.

Nous avons vu que la notion d'unité logique de travail convenait particulièrement à la notion d'exécution de job. Nous suggérons donc l'unité fonctionnelle de gestion d'activité.

Nous avons vu aussi que l'unité logique de travail était décomposée. Dans les exemples précédents, nous avons vu les avantages de la synchronsation mineure pour

la réalisation d'un transfert avec reprise. Pour ces mêmes raisons, nous reprenons les unités fonctionnelles de synchronisation mineure et de resynchronisation.

Le transfert du job et le transfert de son résultat peuvent être alors considérés comme deux unités de dialogue, clairement séparées. Entre ces deux unités aura lieu le dialogue de demande et de réception d'information. Il ne nécessite pas a priori de reprise. On pourrait toutefois séparer ces diverses demandes et réponses, mais nous n'en voyons pas l'intérêt. Nous suggérons donc l'emploi supplémentaire de l'unité fonctionnelle de synchronisation majeure.

Afin de respecter le rapport existant entre les deux utilisateurs, nous ne sélectionnons pas l'unité fonctionnelle de terminaison négociée. Rappelons que l'échange ne nécessite pas de contrôle half duplex. Nous suggérons donc l'emploi de l'unité fonctionnelle full duplex.

Remarquons que le serveur doit pouvoir alerter le maître, de la terminaison du job. Ce message est purement asynchrone par rapport au reste du dialogue. Afin de le différencier, nous suggérons l'emploi de l'unité fonctionnelle de données typées.

Les unités fonctionnelles de données expresses et d'information de capacité peuvent être sélectionnées, de façon optionnelle, pour leurs caractéristiques propres.

- Conséquences sur le traitement d'erreur session.

Le mécanisme de gestion des erreurs est semblable aux deux exemples précédents. Pour toute remarque, le lecteur peut se référer à ces deux exemples.

- Conclusion

Notre sélection reprend donc les unités fonctionnelles suivantes :

- le noyau
- le full duplex
- la gestion d'activité
- la synchronisation majeure

- la synchronisation mineure
- la resynchronisation
- l'envoi de données typées.

Il nous est hélas impossible de comparer notre solution avec celle de la norme, car nous n'avons pas trouvé de documents la reprenant.

5.3.3.2 Spécification 2

Permettre l'exécution de plusieurs jobs à la fois, de demander de l'information sur l'exécution d'un job, de suspendre et de reprendre l'exécution d'un job, d'arrêter un job.

- Etude des notions

- Evolution des notions.

La notion de job est inchangée. Le rapport entre utilisateurs est inchangé. La seule différence réside dans la notion d'exécution de job : plusieurs exécutions sont désormais possibles à la fois.

Cette dernière considération nécessite plusieurs précisions et restrictions.

Nous regroupons logiquement le transfert du job et sa demande d'exécution, et nous considérons ce regroupement comme indivisible. Toutefois, nous admettons que plusieurs demandes de job peuvent se suivre consécutivement, sans attendre les résultats de chaque demande.

Nous considérons aussi comme indivisible et bloquant, le transfert des résultats d'une exécution.

- Conclusions.

Les actions relatives à un même job constituent toujours une unité logique de travail. Les différents transferts gardent leurs caractéristiques.

Une fois le transfert du job et sa demande d'exécution effectués, l'activité pour ce job est interrompue, et laisse la place à d'autres activités.

En dehors d'une activité, on peut demander de l'information sur un job. Cette demande sera considérée comme bloquante, nécessitant une réponse.

Le serveur doit toujours avoir la possibilité de prévenir le maître qu'une exécution est terminée. Cette information asynchrone peut arriver chez le maître, lors d'une activité, ou hors d'une activité. Suite à cette information, le maître, dès qu'il le pourra, reprendra l'activité pour effectuer le transfert des données de résultat, et ensuite, terminera l'activité.

- Choix d'un sous-ensemble

Nous gardons de notre première analyse :

- la sélection des unités fonctionnelles :
  - noyau
  - full duplex
  - synchronisation mineure
  - resynchronisation
  - gestion d'activité
  - données typées
- la non-sélection de l'unité fonctionnelle de terminaison négociée
- la sélection optionnelle de l'unité fonctionnelle de données expresses.

Notons que l'unité fonctionnelle de synchronisation majeure n'est désormais plus nécessaire. En effet, un transfert est, maintenant, une suite de sous-unités de dialogue, clairement séparée des autres par les mécanismes de gestion d'activité. Aucun point de synchronisation majeure ne sépare le transfert d'un job et le résultat de l'exécution de ce job. Afin de ne pas perdre d'information (implication de l'utilisation du SPDU 'prepare'), le dernier point de synchronisation mineur devra être confirmé, avant d'interrompre l'activité.

Dans notre choix d'unités fonctionnelles, nous ajoutons l'unité fonctionnelle d'information de capacité. Elle sera utilisée lors des demandes d'information sur l'état d'exécution d'un job.

5.3.3.3 Conclusion

Notre pseudo JTM, dans sa spécification définitive (spécification 2) nécessiterait donc les unités fonctionnelles suivantes :

- le noyau
- le full duplex
- la gestion d'activité
- la synchronisation mineure
- la resynchronisation
- l'envoi de données typées
- l'envoi d'information de capacité.

Il permet un multiplexage des demandes de job sur une même connexion. Toutefois, une critique importante peut lui être apportée. En effet, lorsque les deux utilisateurs sont "inactifs" (le maître n'a plus de job à transférer et ne désire pas d'information sur les diverses exécutions en cours; l'esclave n'a pas encore de résultat à envoyer) la connexion reste ouverte pour rien. Il serait utile de libérer cette ressource. Il faut cependant éviter que la succession de libérations et de reprises de la connexion n'occasionne d'overhead important sur les performances de transfert. La solution suivante permet de réaliser un compromis entre cette économie et les performances de transfert.

Cette nouvelle solution consiste à permettre au maître de demander la terminaison de la connexion après un certain temps d'inactivité (à déterminer) et de la "réouvrir" (en fournissant lors d'une nouvelle demande de connexion, la même référence de dialogue) par la suite. L'esclave doit pouvoir refuser une demande de terminaison et "réouvrir", lui aussi, la connexion (au cas où il aurait un résultat à transférer). Cette nouvelle solution nécessite l'adoption supplémentaire de l'unité fonctionnelle de terminaison négociée.

5.3.4 Pseudo VT : remote login

- Introduction

L'OSI a créé un modèle pour permettre à un utilisateur d'accéder via un terminal à un ordinateur éloigné : le service de terminal virtuel. [LOW 83] Il a développé une classe de base qui devrait répondre aux besoins d'accés à des applications non-sophistiquées. Cette classe permet entre autres de se connecter, d'envoyer des données.

Nous allons analyser une communication en "remote login" lorsqu'un utilisateur se connecte pour réaliser des commandes d'Operating System (OS) de base.

- Spécification

Notre pseudo VTS doit permettre de se brancher par un service de terminal virtuel, sur un ordinateur éloigné, pour réaliser des opérations de session OS habituelles de base.

- Etude des notions

- Notion d'opération.

Une opération résulte en une commande qui nécessite ou non une réponse (commande d'affichage, commande de contrôle d'affichage). Le transfert de données résultant d'une opération n'est en principe pas très important. Une commande peut être introduite entre l'envoi d'une commande précédente et le transfert de ses résultats. Plusieurs opérations peuvent être demandées consécutivement, une à la fois. Une opération peut être annulée. L'ensemble de ces opérations constitue une session OS de base.

- Rapport entre les utilisateurs.

Le rapport entre utilisateurs est un rapport type maître-esclave. Le maître ouvre une session OS, donne des commandes, et lorsqu'il en a fini, termine la session. L'esclave, toujours appelé, ne peut que répondre. Toutefois, il se peut que l'esclave prévienne le maître d'un événement particulier (message provenant d'un autre utilisateur) de façon asynchrone.

- Conclusions.

Le dialogue est composé d'une seule unité logique de travail. Il n'est donc pas nécessaire d'employer la gestion d'activité. L'unité logique de travail est découpée en sous-unités de dialogue. Toutefois, cette structuration de niveau session n'est pas nécessaire. En effet, le contrôle au niveau session est très limité du fait du rapport existant entre utilisateurs finaux, et de la simplicité du dialogue. Dans ce cas précis, il n'est pas nécessaire de disposer d'outils de reprise du dialogue : en cas de problème, l'utilisateur réagit en envoyant une nouvelle commande.

De plus, le transfert de données de l'esclave est conséquent soit d'une demande du maître qui attend une réponse, soit d'un événement exceptionnel (message provenant système). Le maître doit toujours avoir la possibilité d'envoyer des commandes. Le transfert doit donc pouvoir se dérouler dans les deux sens, sans contrainte de jeton. Nous suggérons donc l'unité fonctionnelle full duplex. Notons que les commandes de contrôle (arrêt d'un processus, de l'affichage) doivent être envoyées et reçues le plus vite possible. De plus, elles ne comportent pas beaucoup de caractères. Nous proposons l'emploi supplémentaire de l'unité fonctionnelle de données expresses.

- Choix d'un sous-ensemble

Vu l'absence de structuration et de contrôle session, nous proposons l'utilisation du sous-ensemble BCS en full duplex, avec en plus l'unité fonctionnelle d'envoi de données expresses.

Hélas, il nous est impossible de comparer notre solution avec celle de la norme, car nous n'avons pas trouvé de document la relatant.

5.4 Conclusion

Notre méthode résulte en une démarche simple qui permet de décomposer le problème du choix en sous-problèmes (choix d'une structuration, d'un traitement d'erreur, de la façon d'envoyer des données et de terminer la connexion) en analysant la structure du dialogue engendrée par l'application et le rapport entre utilisateurs.

Nous insistons sur le fait que cette démarche doit être appliquée avec beaucoup de bon sens. Il faut choisir l'outil qui, de par ses caractéristiques, correspond à la structure de la communication. Il faut être conscient des répercutions sur les performances de l'échange que peut engendrer le choix d'un service de synchronisation (mineure/majeure). Il ne faut pas se servir d'outils quand on n'en a pas besoin (exemple : VTS). Nous tenons aussi à attirer l'attention du lecteur sur la difficulté d'écrire une application réellement full duplex : il vaut mieux soit la ramener à un échange half duplex ("un à la fois"), soit la décomposer en deux applications (exemple : spécification 2 du pseudo X400). En fait, l'utilisateur ne dispose que de son bon sens pour choisir les services qu'il va utiliser et la façon dont il va les utiliser, et pour réaliser un compromis entre cette utilisation, l'utilisation de la connexion, les performances de l'échange, et le nombre de lignes que nécessitera l'écriture du programme d'application (influencé, par exemple, par le finesse d'une analyse d'erreur et du choix du traitement correspondant).

6 Evaluation personnelle

Au premier chapitre, nous avons présenté les concepts mis en jeu par la session. Nous avons souligné un des objectifs essentiels de la norme ISO, à savoir : la généralité. Dans la réalisation de la session, la mise en oeuvre de ses concepts, les concepteurs de la norme ont du opérer des choix. Ces choix ont une influence sur les concepts. De plus, ils remettent en cause la qualité de généralité. Les exemples qui suivent en sont l'illustration. Premièrement, un seul jeton est attribué pour contrôler la structuration en activités et en unités de dialogue. Ceci revient à dire que c'est celui qui structure l'échange en activités qui doit obligatoirement structurer en unités de dialogue (contrairement au rapport entre synchronisation mineure et majeure où les deux "structurateurs" peuvent être différents). Et pourtant, ainsi que nous l'avons démontré, ces deux structurations sont différentes. Deuxièmement, nous avons vu que, dans tous les cas, la structuration engendrée par les mécanismes de gestion d'activité et de synchronisation mineure ne portaient que sur un seul sens du dialogue. En effet, l'addendum sur la synchronisation symétrique ne concerne que la synchronisation majeure. Ceci revient à dire, par exemple, qu'il est possible de structurer les deux flux en unités de dialogue, mais pas en sous-unités de dialogue. Troisièmement, rappelons-nous les liens qui existent entre les jetons et les conséquences qui en résultent sur l'utilisation de certains services (par exemple : dans le cas où seuls le noyau et la synchronisation mineure sont sélectionnées, c'est celui qui détient le jeton de synchronisation qui peut demander la terminaison de la connexion). Enfin, souvenons nous de certains détails tel que la limitation de la taille du paramètre "données de l'utilisateur" des primitives de connexion et de coupure qui en empêche une utilisation optimale.

Les trois exemples cités ci-dessus illustrent aussi la complexité de la session. Cette complexité nuit à la compréhension de ce niveau. Prenons l'exemple de la resynchronisation. Premièrement, l'utilité de ce mécanisme et la procédure d'utilisation de ses primitives varient beaucoup en fonction du choix d'un sous-ensemble particulier. Deuxièmement, il n'est pas évident de comprendre précisément ce que le fournisseur contrôle et ne contrôle pas (il n'y a pas de dépendance entre les unités fonctionnelles de synchronisation et de resynchronisation; mais, lors d'une demande de resynchronisation option redémarrage ou choix de l'utilisateur, la SPM vérifie si au moins une unité de synchronisation a été retenue). Ceci entraîne de grandes difficultés quant à la compréhension du founisseur, à son implémentation et à son utilisation.

Revenons sur le traitement d'erreur. Nous avons déjà insisté sur sa richesse et sa cohérence. Hélas, nous déplorons plusieurs choses : la non-utilisation d'un timer d'inactivité (et sa non-justification) et le "deadlock" qui en résulte, le risque de remise en cause de l'intégrité des unités de dialogue par la resynchronisation option redémarrage, et enfin, le choix des concepteurs quant à la signalisation d'anomalie limitée à un seul sens.

Attardons-nous maintenant sur le fournisseur session. Nous l'avons introduit en énonçant quatre règles générales. A travers le texte, ces règles n'ont souffert que peu d'exceptions : on y retrouve les mécanismes optionnels de segmentation, de concaténation étendue, et l'option "prepare" du protocole. Ces règles sont simples, et induisent un comportement qui l'est tout autant. Toutefois, ce fournisseur réalise des tests qui, à cause de la combinaison des différents services et des conventions prises à ce sujet, sont eux réellement compliqués. Ceci revient à dire que, si un utilisateur ne désire pas utiliser les outils offerts par la session mais plutôt les réaliser au niveau supérieur, il devra faire face à cette réelle difficulté. De plus, il serait à notre avis regrettable de ne pas se servir d'un outil qui, ainsi que nous avons tenté de le démontrer dans l'ensemble du texte, n'implique pas d'overhead important sur l'échange, et tout au contraire, permet de l'optimaliser.

Notre dernière évaluation portera sur les sous-ensembles types : nous allons tenter de les démystifier. Dans la plupart des articles, dès que l'on parle de la session ou de son utilisation, on cite ces trois sous-ensembles. Certains leur voient un avenir prometteur de standard d'utilisation de la session . [RAU 86] Cependant, les concepteurs de la norme citent ces trois sous-ensembles en tant qu'illustrations : ils ne font donc pas partie de la norme en tant que telle. Nous croyons personnellement que leur existence en tant que standards d'utilisation ne se justifie que s'ils sont réellement utilisés comme tels par d'autres normes, ou bien que, de par leur composition, ils concernent un ensemble précis d'applications qui sont susceptibles de les référencer. De notre analyse, il ressort que BCS peut servir de référence : c'est le sous-ensemble minimal proposable par une application half ou full duplex. Il concerne donc l'ensemble des applications qui ne désirent (ou ne nécessitent) pas ou très peu de contrôle de niveau session. BAS sert lui de référence : il existe un rapport intime entre ce sous-ensemble et la norme T62, ainsi qu'avec X400. Toutefois, nous nous sommes étonnés de sa caractéristique "Basic" : en effet, la structuration qu'il offre est déjà très évoluée (unités logiques de travail décomposées en sous-unités de dialogue). Quant à BSS, dans l'état actuelle de nos investigations, il ne sert pas de référence. Rappelons-nous les nombreuses questions

posées lors de l'étude de sa composition, et la conclusion tirée en ce qui concerne son utilisation par des applications : à notre avis, il ne concerne qu'un ensemble très limité d'applications. Nous nous étonnons aussi de sa caractéristique "Basic" : en effet, il permet de structurer l'échange en unités de dialogue composées de sous-unités de dialogue. Nous nous sommes alors posés la question suivante : BSS n'existerait-il pas relativement aux autres sous-ensembles types ? Hélas, de notre analyse, il ressort que ces trois sous-ensembles ne partitionnent ni l'ensemble des applications utilisatrices, ni l'ensemble des services session. Nous nous demandons dès lors sur quoi se basent certaines personnes lorsqu'elles voient pour BSS, un avenir prometteur de standard d'utilisation. A notre avis, ce sous-ensemble très particulier ne se justifie seulement qu'en tant qu'illustration.

7 Conclusion

L'objectif de ce chapitre était d'analyser en profondeur les services offerts par la couche session, d'en souligner les caractéristiques afin d'en dégager des critères d'utilisation, et d'établir une démarche facilitant le choix d'unités fonctionnelles pour une application particulière.

Pour ce faire, nous avons d'abord analyser les services et les liens qui les unissaient. Ensuite nous avons regroupé et présenté les services concernant le traitement d'erreur. Nous avons alors étudié les sous-ensembles types et déterminé à quelles types d'applications ils étaient destinés. Fort de ces différentes analyses, nous avons proposé et appliqué une démarche de choix d'un sous-ensemble correspondant à une application particulière.

De toute cette étude, on constate que la réalisation de la session a des répercussions sur ses concepts ainsi que sur la qualité de généralité de cette couche. De plus, la mise en oeuvre de ses concepts est assez complexe, ce qui peut nuire à sa compréhension et à son utilisation.Toutefois, la structuration, le traitement d'erreur et les possibilités d'envoi de données sont riches. Nous en concluons que la session est en général un bon outil d'organisation et de synchronisation d'un échange de données qui, de surcroît, n'implique pas un overhead important sur le transfert : il est donc, à notre avis, intéressant de l'utiliser.

Pour un utilisateur désireux de se servir de cette couche, le choix d'une combinaison d'unités fonctionnelles ne se limite pas aux trois sous-ensembles types : ils ne constituent ni une partition des services, ni une partition de l'ensemble des applications utilisatrices. Ces trois sous-ensembles servent tantôt d'illustration, tantôt de référence. Le choix d'un sous-ensemble particulier étant un problème assez complexe, l'application de notre démarche (qui elle est très simple) peut faciliter la tâche des futurs utilisateurs. Dans l'application de cette démarche, le bon sens est la qualité essentielle dont ils devront faire preuve.

IV CHAPITRE 3 : ANALYSE D'UNE IMPLEMENTATION

1 Introduction

Dans la première partie, nous avons procédé à une approche purement théorique basée presqu'exclusivement sur les textes des différentes normes. Afin de montrer que l'approche de l'ISO n'appartient pas qu'au monde de la théorie et de découvrir les problèmes soulevés par une expérience pratique, nous avons cherché une implémentation des protocoles des niveaux supérieurs. Cette implémentation, "nous" l'avons trouvée : elle s'appelle ISODE.

ISODE est une implémentation des couches 5, 6 et 7 orientées connexion. Les services de niveau 4 sont implémentés, mais le transport est assuré par TCP/IP. Au dire même de ses concepteurs, ce n'est pas un "production software". Il s'agit d'un ensemble de programmes dont le but est de permettre le développement rapide d'applications reposant sur le modèle ISO. Nous lui avons trouvé d'autres intérêts. En étant une réalisation particulière, ISODE permet de mieux comprendre des mécanismes généraux tels que l'adressage, l'initialisation du "stack" ISO, et l'interface. De plus, il offre la possibilité d'approfondir ses connaissances théoriques sur un niveau particulier. Enfin, de par son architecture, il permet de développer une couche de conception personnelle ou une application particulière reposant, par exemple, directement sur le niveau 5.

Dans ce chapitre, nous allons d'abord présenter ISODE de façon générale par l'établissement d'une "fiche technique", l'explication de son architecture et la présentation générale de son initialisation lors de l'établissement d'une connexion. Ensuite nous analyserons l'implémentation de la couche session en présentant son architecture logique et en insistant sur trois points qui nous paraissent importants (l'interface, l'adressage et l'initialisation). Nous analyserons alors les choix des concepteurs, et nous étudierons la conformité de cette réalisation par rapport à la norme ISO. Afin d'illustrer nos propos, nous présenterons enfin une petite application de transfert de fichiers reposant directement sur la couche session. Pour terminer, nous apporterons une évaluation personnelle sur ISODE en général et sur l'implémentation de la session en particulier.

L'exposé qui va suivre est basé principalement sur l'analyse du texte des programmes sources, sur l'étude du manuel utilisateur et accessoirement, sur quelques tests et implémentations personnelles.

2 Présentation générale d'ISODE

2.1 Fiche technique

ISODE a été réalisé par Marshall T. Rose et Dwight E. Cass au Northrop Research and Technology Center. C'est une réalisation des 4-, 5-, 6- et 7-SAP (Service Access Point) tels que définis par l'ISO. Les protocoles de niveau 5 et 6 ont été implémentés. Le niveau 7 contient trois Common Application Service Element (CASE) : l'Association Control Service (ACS) qui permet d'établir et de rompre une association entre applications, le Remote Operation Service (ROS) qui permet de demander l'invocation d'opérations à distance (service de type client/serveur) et le Reliable Transfer Service (RTS) dont nous avons déjà parlé. ISODE utilise le "DoD Transmission Control Protocol" (TCP) au lieu du protocole ISO pour fournir les services de niveau 4. Une version ultérieure contiendra une implémentation de FTAM. Le langage de programmation utilisé est le C. Les programmes sont installables sur mini-ordinateurs dotés d'un des systèmes d'exploitation suivant : Berkeley Unix, AT&T Unix, HP-Unix et ROS. Ils le seront prochainement sur PC.

La version que nous avons étudiée est la 2.0. Nous l'avons installée sur un VAX avec comme système d'exploitation UNIX 4.2 BSD. Notons qu'au point de vue performance de communication, d'aprés ses concepteurs, l'utilisation des programmes de cette version (service ROS, RTS) diminue le taux de transfert de données obtenu à la frontière de TCP, en moyenne de 12%. Au point de vue espace mémoire, l'ensemble des librairies opérationnelles occupe 1 Mega d'octets. La session en prend 8,5% (comparativement, la couche présentation en occupe 11,7% , le RTS 6,1%).

La "release 2.0" est décomposée en 18 directories contenant en tout plus de 600 fichiers. Pour ne citer que les plus intéressantes directories, on y retrouve : une directory par implémentation d'un SAP, une directory spéciale comprenant un manuel de l'utilisateur assez complet. Notons que quelques programmes de test et de démonstration sont éparpillés dans l'ensemble.

La configuration, la génération et l'installation se réalisent automatiquement. La configuration revient à préciser sur quel type d'OS et sur quelle organisation particulière de fichiers système ISODE va être installée. La génération provoque la compilation de l'ensemble des programmes, conditionnellement à l'OS choisi. L'installation consiste principalement à placer les fichiers exécutables.

2.2 Architecture

2.2.1 Présentation de l'architecture

Les concepteurs ont désiré adopter une architecture de programme qui se veut strictement semblable à celle du modèle de référence. La figure IV.1 illustre cette architecture.

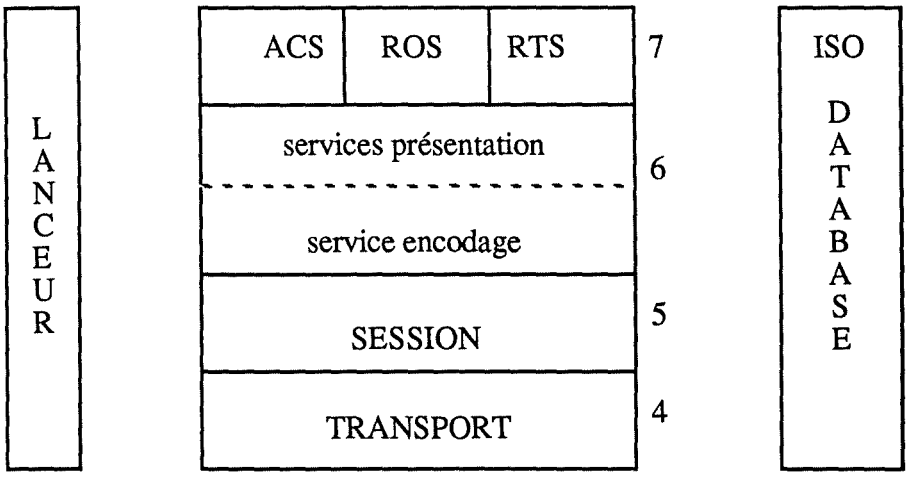


Fig IV.1 : architecture d'ISODE

Le niveau 4 utilise les services de TCP/IP pour offrir globalement un service transport. Le niveau 5 utilise les services de niveau 4 pour offrir les services session. Le niveau 6 est éclaté en deux sous-couches : la première s'occupe de l'encodage des structures de données; la seconde implémente le service présentation. Le niveau 7 comprend les trois CASE : ACS utilise le niveau 6, RTS et ROS utilisent soit les services de niveau 6, soit directement les services de niveau 5. Les adresses des services et des entités se trouvent dans l'ISO DATABASE. Le lanceur est le module qui est chargé lors d'une demande de connexion réseau, d'initialiser le "stack" complet, jusqu'à l'application.

2.2.2 Relations entre couches

Afin d'illustrer les relations entre couches, nous allons maintenant expliquer comment, avec ISODE, il est possible d'établir une association entre deux applications. Suite à cela, nous verrons comment l'ensemble des couches s'initialise, chez l'appelant et l'appelé.

ISODE permet d'établir une communication entre deux applications de cinq façons différentes. La façon la plus standard est d'utiliser les services d'ACS : ce CASE est expressément dédié à l'établissement de pareille connexion. De plus, il respecte le schéma standard ISO (il établit une connexion de niveau 7 en se servant des services de niveau 6). Le RTS et le ROS permettent aussi d'établir une connexion de façon standard (en utilisant les services présentation). De plus, ils permettent l'établissement d'une connexion "façon X400", en utilisant directement les services de niveau 5. L'utilisation directe des services de niveau session provient du fait que X400 définit une syntaxe abstraite de niveau présentation ainsi que le codage des structures de données à utiliser : il n'est donc pas nécessaire de négocier les caractéristiques de niveau présentation en utilisant la procédure de connexion (de niveau 6) dédiée à cet effet.

Voyons maintenant comment le "stack" ISODE s'initialise, suite au lancement d'une application. Nous supposons, pour rester standard, que cette application utilise ACS. Dans le programme qui réalise cette application, on retrouve un appel à la primitive de connexion ACS. Cette primitive fait appel à la primitive de niveau inférieur et ainsi de suite. C'est ainsi que la demande de connexion va descendre dans le stack ISO. Dans sa réalisation pratique, ces diverses primitives sont représentées sous forme de fonctions. Ces fonctions sont compilées et réparties dans des librairies, par niveau. L'activation de la primitive de connexion de niveau 4 entraîne une demande de communication réseau. Cette demande arrivera chez l'appelé à la frontière de TCP/IP (interface socket) et de la couche transport. Là, elle sera captée par un programme particulier : le démon. L'objectif de ce programme est d'écouter les événements à la frontière de TCP/IP, de saisir les demandes de connexion et de créer, par demande, un processus lanceur. Ce lanceur va faire appel consécutivement aux primitives d'initialisation des diverses couches. Pour chaque couche, il sélectionne l'entité adéquate, en fonction de l'adresse de ce niveau fournie par l'appelant et des renseignements contenus dans l'ISO DATABASE. C'est ainsi que la demande de connexion remonte le stack ISO jusqu'à l'application appelée.

En résumé, le lancement d'un programme application (l'appelant) entraîne la création d'un processus qui fera appel à des fonctions (primitives) réparties dans des librairies, par niveau. Du côté appelé, c'est le démon qui crée un processus lanceur qui, une fois l'initialisation terminée, passe la main à l'application appelée. Tout comme l'appelant, cette dernière application fera appel à des fonctions réparties dans les diverses librairies. Un CASE utilisé de façon standard fait appel aux fonctions présentation réparties dans la librairie de niveau 6. Un CASE utilisé "façon X400" fait appel aux fonctions de la librairie de niveau 5.

3 Analyse de la session ISODE

Notre analyse de la session ISODE se compose de cinq sections. La première comprend une présentation générale d'une couche telle que conçue par Marshall T. Rose et Dwight E. Cass, en l'occurrence la session. La deuxième est consacrée à l'interface, la troisième à l'adressage et à l'initialisation. La quatrième est une analyse des autres choix d'implémentation plus particuliers à la session. Enfin, la cinquième est une analyse de conformité à la norme ISO.

3.1 Présentation générale de la couche session ISODE

La couche session ISODE est une implémentation des normes de niveau 5 que nous avons analysées dans les chapitres précédents : elle offre un service orienté connexion et ne permet pas de synchronisation symétrique.

L'architecture de cette couche est illustrée à la figure IV.2.

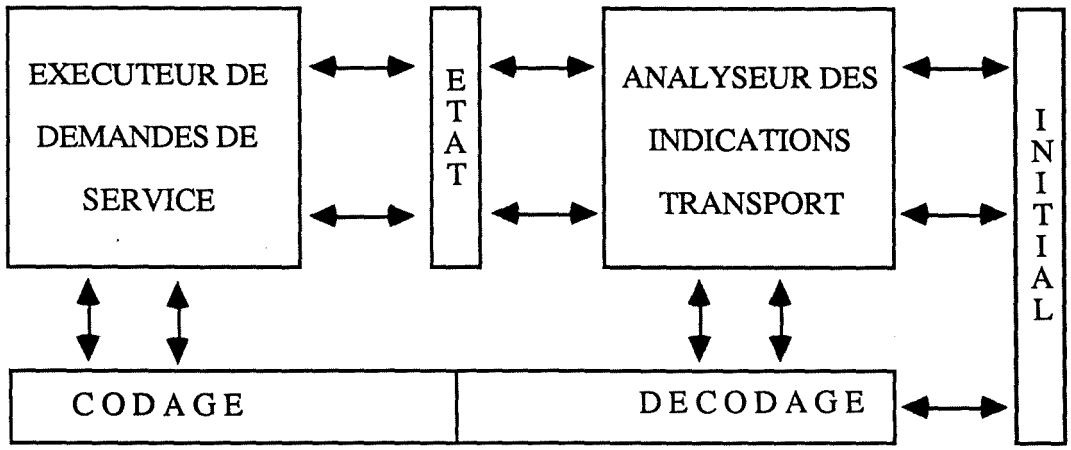


Fig IV.2 : architecture de la couche session

Nous avons appelé le premier module "exécuteur de demande de service" : c'est lui qui est chargé, lorsqu'un niveau supérieur demande un service (requête, réponse), d'exécuter cette demande, en fonction de l'état actuel de la SPM, grâce aux primitives de codage, et en utilisant les services transport. Les primitives de requête et de réponse sont écrites sous forme de fonctions de librairies appelées par le niveau supérieur. Elles comprennent les paramètres décrits dans la norme.

Le module "analyseur des indications transport" est chargé d'analyser tout ce qui est passé de la couche transport à la couche session : lorsqu'arrive une indication transport, il la décode, en extrait le SPDU, l'analyse en fonction des variables d'état, et prépare l'indication (ou la confirmation) de service correspondante. Les primitives d'indication et de confirmation sont exprimées sous forme de structure de données, consultables par les niveaux supérieurs. Remarquons toutefois que trois indications transport échappent aux règles générales citées ci-dessus : l'indication de connexion transport (elle est saisie par le lanceur qui se charge lui-même d'y répondre), la confirmation de connexion transport (saisie par l'exécuteur des demandes de service suite à laquelle il envoi le SPDU de connexion), l'indication de déconnexion transport quand elle est reçue suite à la procédure normale de déconnexion (la session est déconnectée normalement, la SPM retourne instantanément à l'état repos).

Les deux premiers modules n'entretiennent de relations que par l'intermédiaire du module "'état" dont ils lisent et mettent à jour les variables suite aux événements de frontière (4-5, 5-6). Ce module comprend les variables dont les valeurs reflètent l'état de la SPM. Le module "codage/décodage" contient les procédures chargées de coder et décoder les SPDU et les TSDU. Le dernier module, "INITIAL", est celui qui est chargé, lors d'une indication de connexion transport (simulée par le lanceur), d'initialiser la couche.

3.2 Interface entre une entité session et son entité utilisatrice

L'interface est la frontière qui sépare deux couches. A cette frontière se passent des événements (requête, indication, réponse, confirmation). Dans le langage de l'ISO, le 5-SAP est l'interface logique entre une entité session et son entité utilisatrice. [DAY 83] Cet interface peut être réalisé de deux façons, synchrone ou asynchrone. Un interface entre une entité "utilisateur" et une entité "fournisseur" est dit synchrone lorsque c'est l'utilisateur qui demande au fournisseur s'il s'est passé un événement. Il est dit asynchrone lorsque le fournisseur peut, à n'importe quel moment, interrompre son utilisateur pour lui signaler l'occurrence d'un événement. Remarquons que le type d'interface concerne principalement les indications et les confirmations.

ISODE permet pour la phase de transfert (après l'établissement de la connexion) de définir un interface synchrone ou asynchrone. Dans le cas d'un interface synchrone, l'utilisateur dispose d'une fonction de lecture qui lui permet, quand il le désire et aussi longtemps qu'il le désire, de demander au fournisseur si un événement s'est produit. Dans le cas d'un interface asynchrone, l'utilisateur doit préciser à son fournisseur, pour chaque indication et confirmation, quel sous-programme (procédure, fonction) de son niveau traite cet événement. Le fournisseur, dès qu'un événement se produit, active le sous-programme utilisateur adéquat.

Les utilisateurs de la couche session ont donc le choix, pour la phase de transfert, entre ces deux types d'interface. Ce choix est un problème difficile qu'il n'est pas aisé de résoudre complètement de manière tout à fait générale. Nous n'avons pas trouvé de critère pertinent de choix. Le rapport entre utilisateurs "paires" ne nous est pas paru déterminant. Dans le cadre d'un rapport maître-esclave, le maître envoie des commandes et attend des réponses; l'esclave lui attend les commandes pour pouvoir y répondre. Dans les deux cas, l'attente peut se dérouler de façon synchrone ou asynchrone. Un critère de choix peut être alors la facilité d'implémentation (cfr illustration). Dans le cadre d'un rapport symétrique, à tout moment, les deux utilisateurs peuvent émettre et recevoir quelque chose. Nous ne voyons pas comment l'un ou l'autre type d'interface pourrait résoudre simplement cette situation.

3.3 Adressage et initialisation de la couche session

Examinons maintenant comment les concepteurs d'ISODE ont résolu le problème de l'adressage et de l'initialisation au niveau de la session. Nous allons d'abord nous intéresser au côté appelant, ensuite au côté appelé.

Du côté appelant, l'utilisateur (présentation ou application) active la primitive d'établissement de connexion session. Il fournit entre autres comme paramètres l'adresse du 5-SAP appelant et appelé. L'adresse du 5-SAP appelant permet à l'utilisateur d'accéder à l'entité session désirée, et au fournisseur, de déduire le 4-SAP qu'il va employer. Dans ISODE, vu qu'il n'y a qu'une seule entité session et une seule entité transport (et donc, un seul 4-SAP), ce paramètre n'est d'aucune utilité. En fait, la primitive session de connexion, lorsqu'elle est activée, prépare les paramètres de la primitive de connexion de niveau inférieur, et ensuite, active cette primitive.

Du côté appelé, nous avons vu qu'un progamme particulier créait un lanceur. Ce lanceur est chargé d'activer tour à tour les procédures d'initialisation (INITIAL) des différentes couches. En ce qui nous concerne, une fois que la couche transport est initialisée, le lanceur reprend la main. Il active "l'initialisateur" session adéquat (dans ISODE, il n'y a qu'une entité session et donc, un seul "initialisateur"). "L'initialisateur" est chargé de recevoir le SPDU de connexion et de préparer une indication de connexion. Le lanceur reprend alors la main. Avec l'adresse du 5-SAP appelé, il consulte l'ISO DATABASE afin de découvrir quelle entité supérieure il doit activer : soit le niveau présentation standard, soit un module application (façon X400). L'ISO DATABASE contient en fait la correspondance entre un (N)-SAP et la (N+1)-entité qui l'utilise (principe de la (N+1)-directory). [DAY 83] Une fois cette information connue, le lanceur active "l'initialisateur" adéquat qui répondra à la demande de connexion session. L'initialisation de la couche session sera alors terminée.

3.4 Présentation des choix particuliers à la session

Nous avons vu que la session ISODE est orientée connexion, qu'elle permet la réalisation d'un interface synchrone ou asynchrone, qu'elle peut être utilisée par la couche directement supérieure ou par une autre couche. De plus, nous avons détaillé son architecture interne. Analysons maintenant les choix d'implémentation plus particuliers au founisseur session.

Au deuxième chapitre, lorsque nous parlions de la négociation, nous avons introduit la notion de Quality Of Service (QOS) qui regroupe certaines caractéristiques de connexion. Dans ISODE, ces paramètres ne sont pas négociables. En effet, le fournisseur ne réalise pas de calcul de taux d'erreur résiduel, de débit, de temps de transit, et n'assure ni de priorité, ni de protection de connexion. Il ne permet pas non plus la concaténation étendue (impliquée par la demande du paramètre QOS "transfert avec optimisation du dialogue"). Par contre, le contrôle étendu est toujours demandé par la SPM.

Nous avions abordé le problème de la décision, chez l'appelé, d'accepter les paramètres de l'indication de connexion session proposés par l'appelant. Nous avons vu que, en toute logique, cette décision revient au niveau application. Hélas, lorsque l'on suit le modèle de référence de façon standard, cette décision revient au niveau présentation qui lui ne possède pas les éléments nécessaires à pareille décision. Dans ISODE, lorsque cette décision revient à la présentation, les propositions de l'appelant sont directement acceptées : dans ce cas, il n'y a plus de négociation des caractéristiques de communication de niveau 5.

Le traitement d'erreur est réalisé assez simplement. En ce qui concerne les erreurs d'utilisation des services session (type IV), les concepteurs ont créé un mécanisme particulier qui informe l'utilisateur de la première erreur que le fournisseur détecte (valeur de paramètre, erreur de procédure) lors d'une demande de service : il laisse ainsi l'initiative du traitement à l'utilisateur. En ce qui concerne les erreurs de niveau 5 (type I : erreur de protocole session), le service de signalisation d'anomalie par le fournisseur n'a pas été implémenté : ceci revient à dire que, à la moindre erreur de protocole, le fournisseur coupe la connexion session.

Notons encore deux choix de détail. Premièrement, ainsi que nous l'avons introduit au chapitre 2, les concepteurs trouvent la taille du paramètre données de l'utilisateur de la primitive de coupure par le fournisseur, anormalement petite (9 octets). Ils ont dès lors décidé de la faire passé à 512 octets. Notons toutefois que ce choix délibéré n'est pas autorisé par la norme. Deuxièmement, nous avons aussi introduit au chapitre 2 la possibilité laissée au fournisseur de garder ou de couper une connexion transport suite à une déconnexion session (coupure ou terminaison). Les concepteurs ont décidé que, lorque la connexion session était terminée, on abandonnait la connexion transport.

3.5 Etude de conformité

Nous allons, dans cette section, exposer notre étude de conformité de l'implémentation de la session ISODE aux normes session ISO. Nous tenons à signaler que nous n'avons pas analysé le codage et le décodage des SPDU : leur implémentation est simple et nous l'avons supposée correcte. Notre étude ne concerne donc que les deux modules principaux, à savoir l'exécuteur de demandes de service et l'analyseur des indications transport.

Le tableau IV.1 reprend, pour le module exécuteur de demandes de service, par primitive, les erreurs que nous avons détectées. Pour les primitives concernées, nous mettons en rapport les tests effectués par ISODE, et ceux cités dans la norme. Notons que, lorsque ces tests ne sont pas respectés, on se retrouve normalement face à une erreur locale d'utilisation de service (type IV).

PRIMITIVES	TESTS ISODE	TESTS ISO
SConnReq.	SI UF (signalis. anomalie) ALORS UF (half duplex)	SI UF (signalis. anomalie) ALORS UF (half duplex) SI UF (informat. capacité) ALORS UF (gestion activité)
SMinSyncResp.	NSS < = V(A)	V(M) < NSS < = V(A) Vact = vrai
SResyncReq. choix de l'util. redémarrage		soit UF (mineure) soit UF (majeure)
SActIntrReq. SActDiscReq.	interdits si soit : attente confirm. majeure fin activité resynchron.	permis dans ces cas

UF (xxx) : l'unité fonctionnelle xxx a été sélectionnée
NSS : numéro de série de synchronisation
V(A) : plus petit numéro de synchronisation à confirmer
V(M) : prochain numéro de synchronisation à utiliser
Vact : variable indiquant si l'on se trouve en activité

Tableau IV.1 : irrégularités concernant l'exécuteur de demandes de service

SPDU	TESTS ISODE	TESTS ISO
demande connex.	SI UF (signal. anomal) ALORS UF (half duplex)	SI UF (signal. anomal) ALORS UF (half duplex) SI UF (informat. capacité) ALORS UF (gestion activité)
pose majeur	NSS = V(M)	NSS = V(M) Prédicat (activité) Prédicat (jeton)
demande fin act.	NSS = V(M)	NSS = V(M) Prédicat (activité) Prédicat (jeton)
pose mineur		NSS = V(M) Prédicat (activité) Prédicat (jeton)
réponse mineur	V(A) <= NSS < V(M)	V(A) <= NSS < V(M) Prédicat (activité) Prédicat (mineur)
demande resync. option redémar.	NSS <= V(R)	NSS <= V(R) Prédicat (activité)
anomalie fournis.		Prédicat (activité) Prédicat (jeton)
demande anomal.		Prédicat (activité)
début activité		Prédicat (jeton)
reprise activité		Prédicat (jeton)

UF (xxx) : l'unité fonctionnelle xxx n'a pas été sélectionnée
NSS : numéro de série de synchronisation
V(M) : prochain numéro de synchronisation à utiliser
V(R) : plus petit numéro de resynchronisation redémarrage
V(A) : plus petit numéro de synchronisation à confirmer
Prédicat (activité) : si l'unité fonctionnelle de gestion d'activité a été sélectionnée, on est en activité
Prédicat (jeton) : vérification de la non-attribution des jetons
Prédicat (mineur) : vérification du droit d'émettre une confirmation de mineur

Tableau IV.2 : irrégularités concernant l'analyseur des indications transport

Le tableau IV.2 reprend par SPDU, les anomalies que nous avons relevées pour le module analyseur des indications transport. Pour chaque SPDU concerné, nous mettons en rapport les tests effectués dans ISODE, et ceux cités dans la norme. Notons que, lorsque ces tests ne sont pas respectés, on se retrouve normalement face à une erreur de protocole session (type I). Dans ce cas, le fournisseur soit signale une anomalie, soit coupe la connexion. Remarquons de plus que, lorsque les tests ISODE (incomplets) concernant la réception de certains SPDU (pose de point de synchronisation majeure, demande de fin d'activité, réponse à une pose de point mineur, demande de resynchronisation option redémarrage) ne sont pas respectés, le fournisseur se contente de "laisser tomber" ces SPDU : il n'en suit ni l'activation d'un traitement d'erreur, ni l'indication correspondante.

Dans le module des demandes des primitives de service, on retrouve peu d'erreurs : elles concernent principalement la structuration et la resynchronisation, et résident dans les tests. Cela provient d'après nous, du fait que les concepteurs, peut-être par souci d'économie d'écriture, n'ont pas repris la présentation de la norme qui, pour chaque événement, donne les tests et les actions à réaliser. Ils ont préféré regrouper des traitements, ce qui les a amenés à tantôt éclater, tantôt regrouper certains tests. C'est dans le module analyseur des indications transport que l'on retrouve le plus d'irrégularités (voir paragraphe précédent). Cela provient, à notre avis, du fait que les concepteurs n'ont pas respecté le principe suivant du protocole : les tests exécutés par la SPM expéditrice sur une demande de service, sont réitérés par la SPM qui reçoit le SPDU correspondant à la demande de service. Si ce "surcontrôle" peut paraître excessif, il est toutefois obligatoire. Il permet en fait de détecter le mauvais fonctionnement d'une SPM par la SPM avec laquelle elle correspond. Hormis ces quelques erreurs, l'implémentation nous paraît respecter la norme ISO.

4 Illustration

Afin d'illustrer l'ensemble de nos propos, nous avons réalisé et installé une petite application de transfert de fichiers reposant directement sur le niveau 5. Nous allons maintenant présenter cette application ainsi que sa réalisation. Les programmes (en pseudo langage et en langage C) se trouvent dans les annexes.

Notre application permet de transférer un fichier de façon synchronisée entre deux systèmes analogues. Elle est réalisée par deux entités, un client et un serveur, qui utilisent directement les services session ISODE (pas de service présentation). Le client fait appel à son serveur et lui transmet le nom du fichier. Le serveur teste l'existence de ce fichier sur son système. Si le fichier existe sur le système appelé, le client clôture l'échange; sinon, il transfère le fichier.

Le schéma de la figure IV.3 est un diagramme d'état qui représente le comportement du client. Ce dernier se trouve premièrement à l'état repos. Suite à l'envoi de la demande de connexion et à la réception de l'acceptation (phase de connexion), il entre en phase 1. Là, il envoie le nom du fichier qu'il désire transférer et reçoit la réponse du serveur. Dans le cas où la réponse lui indique que le fichier existe sur le système appelé, il se déconnecte. Sinon, il entre en phase 2. Tant qu'il y a des pages dans le fichier, il envoie une page, un point de synchronisation et en attend la confirmation. Une fois le transfert terminé, il entre dans la phase de déconnexion. Le schéma de la figure IV.4 représente lui le comportement du serveur : il est symétrique à celui du client.

Le traitement d'erreur que nous avons adopté est très simple : dans le cas d'une erreur d'utilisation des services session, ou d'une erreur de protocole session, ou encore une erreur de protocole de niveau application, on coupe la connexion; dans le cas d'un message incompris, on resynchronise au dernier point de synchronisation confirmé, et on reprend le dialogue en ce point.

Afin de structurer le transfert des pages, nous utilisons les services de synchronisation mineure. Pour respecter le rapport entre utilisateurs et jouïr du contrôle half duplex, nous sélectionnons l'unité fonctionnelle half duplex. Pour permettre au serveur de répondre après le transfert du nom du fichier, nous reprenons le service de données typées.

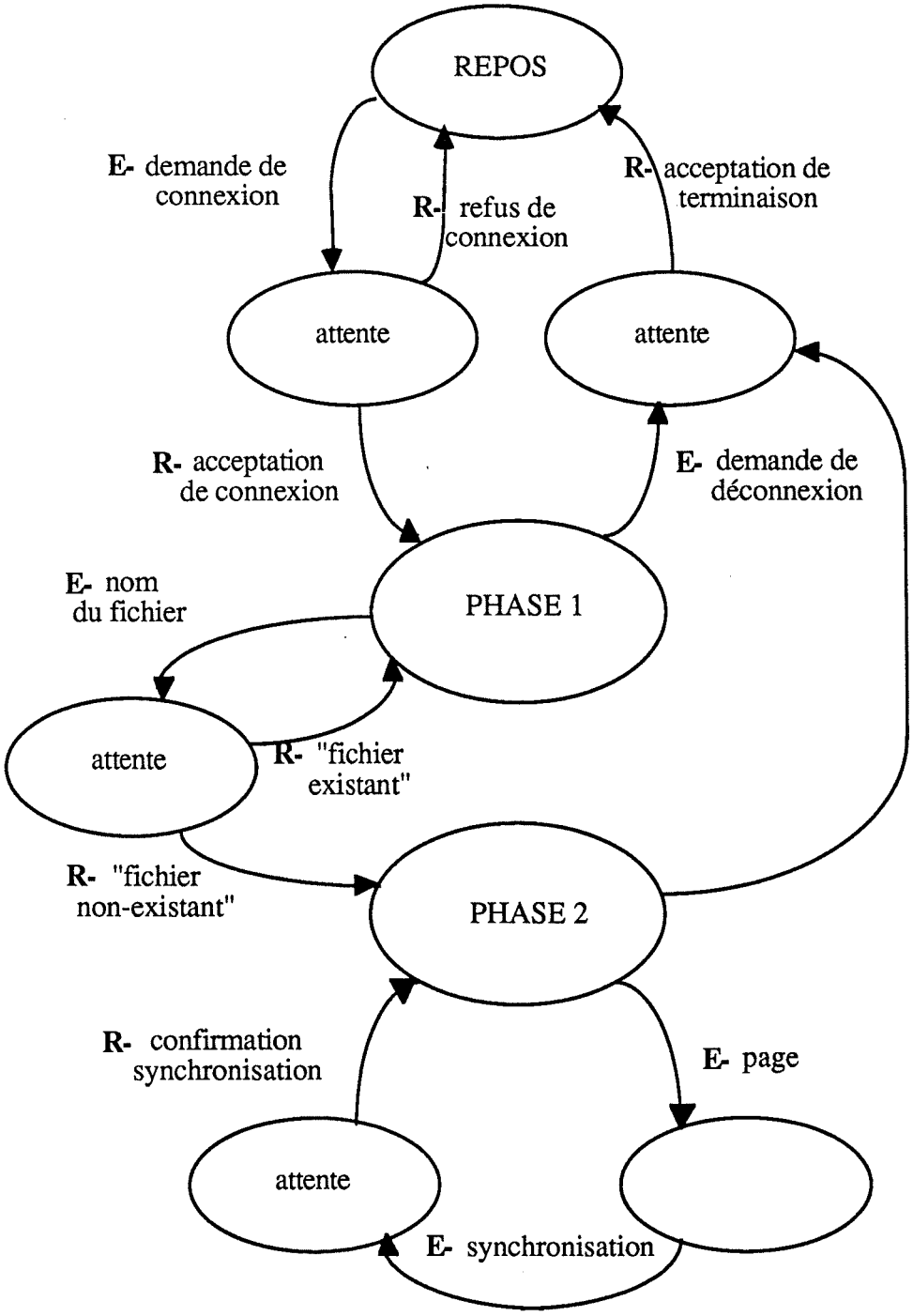


Fig IV.3 : comportement du client

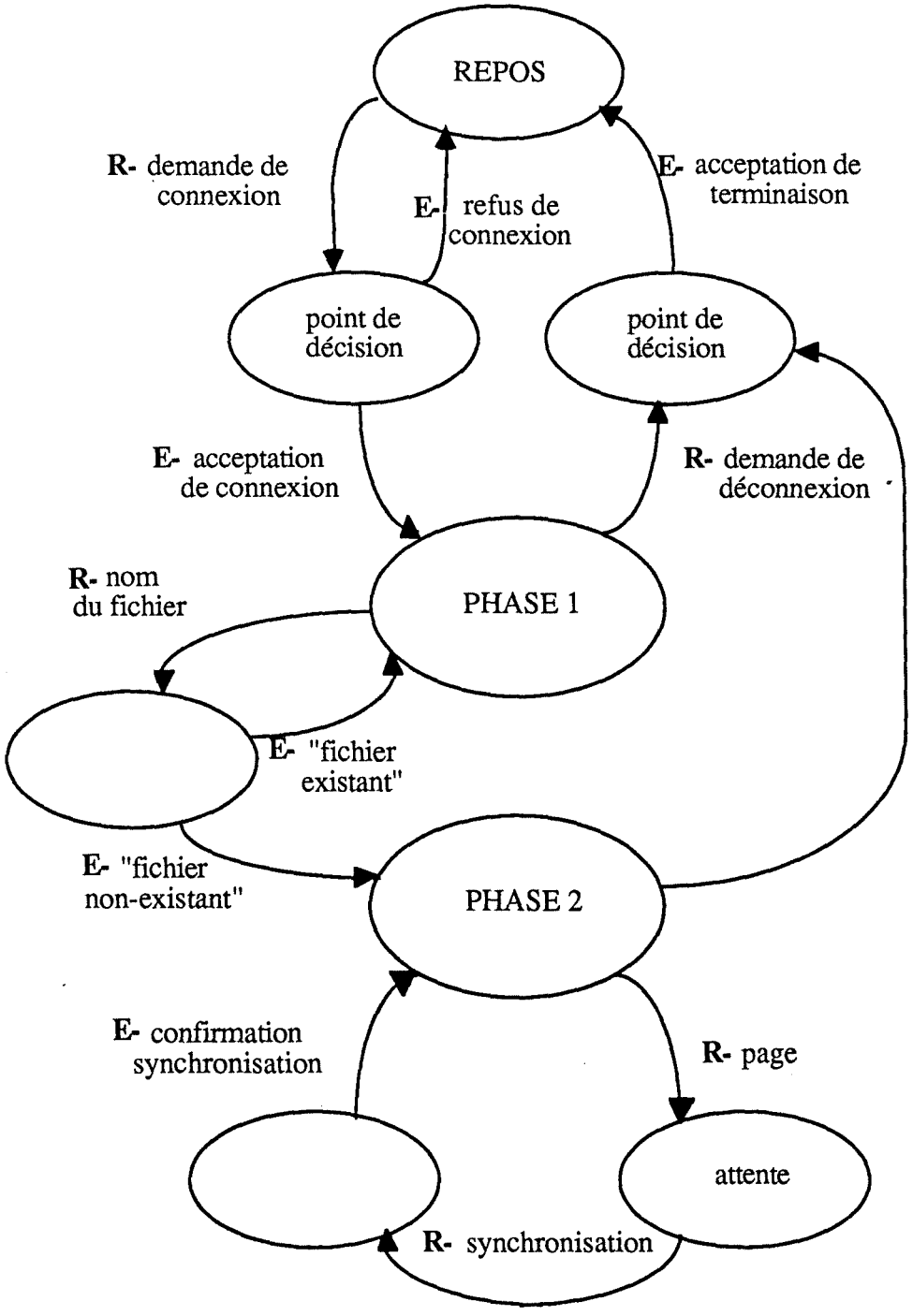


Fig IV.4 : comportement du serveur

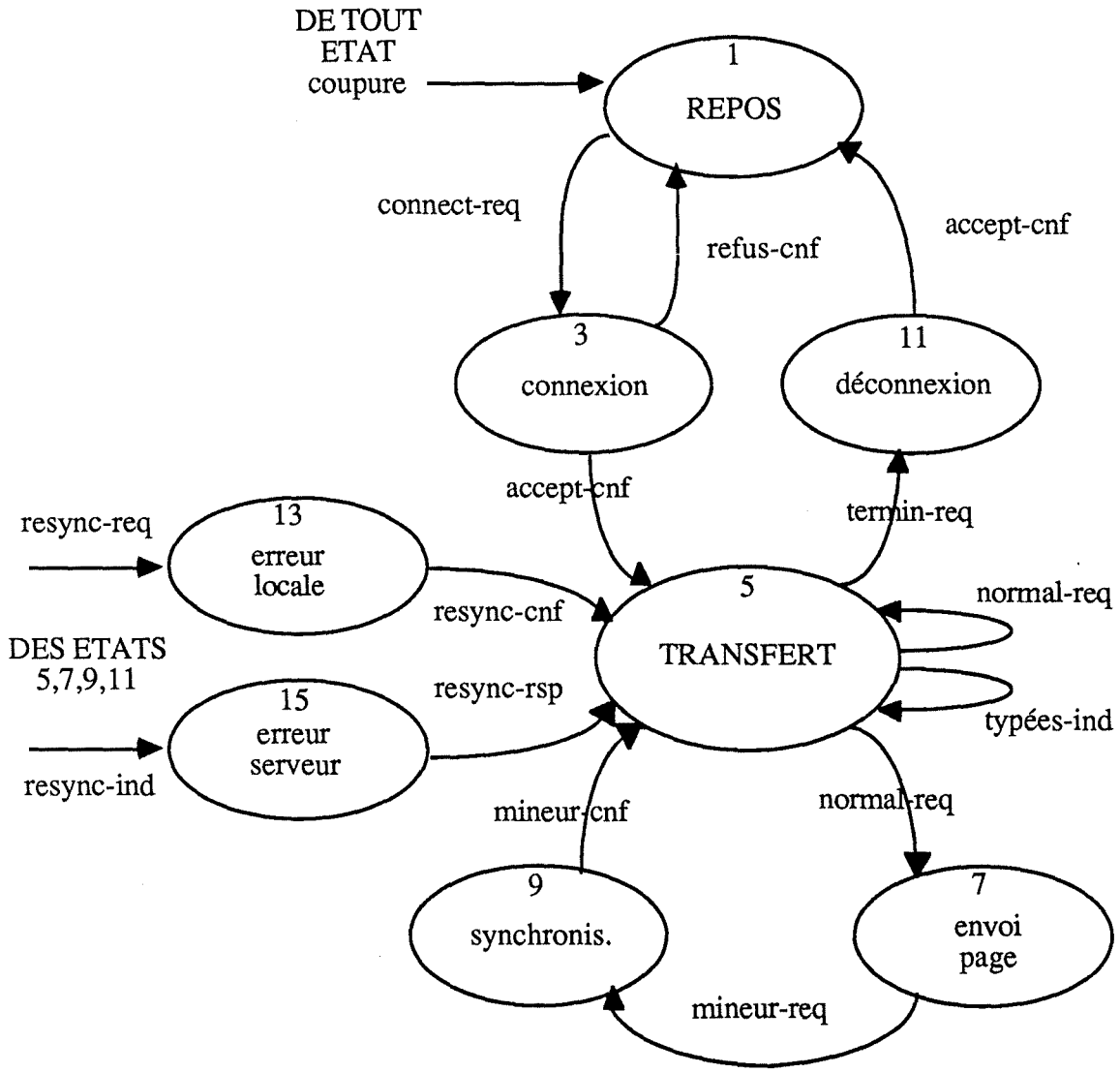


Fig IV.5 : utilisation des services session (client)

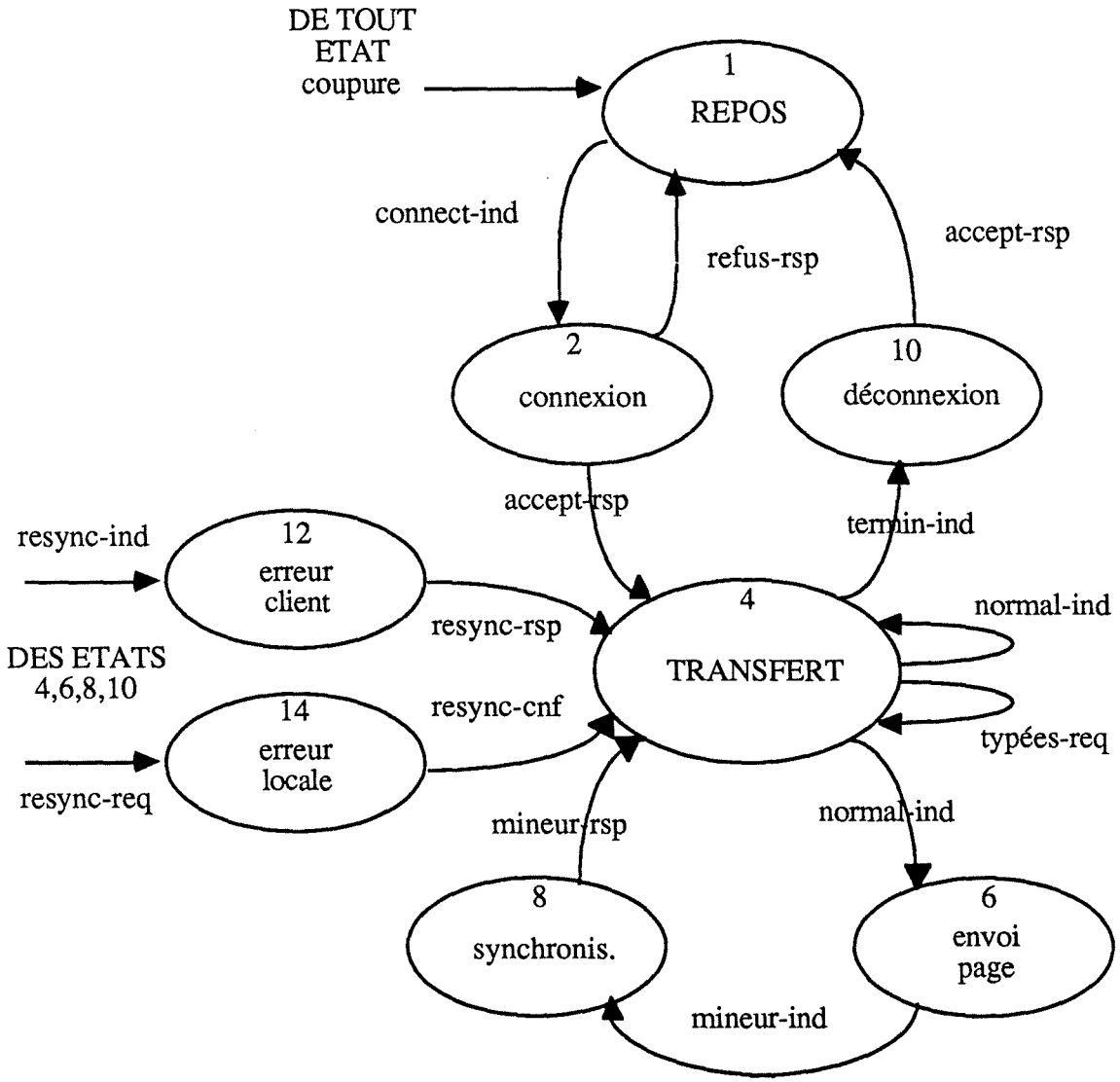


Fig IV.6 : utilisation des services session (serveur)

La figure IV.5 reprend l'utilisation faite par l'entité client des services de niveau session. Le passage de l'état 1 à l'état 5 correspond à la phase de connexion, et celui de l'état 5 à l'état 1, à la phase de déconnexion. L'envoi du nom du fichier (primitive de données normales) et la réception de la réponse du serveur (primitive de données typées) se déroulent dans l'état 5 (partie droite de l'état transfert). Le transfert du fichier page par page (primitive de données normales) est exprimé par le passage successif dans les états 5, 7, 9, 5. Pour les services de traitement d'erreur, le graphe spécifie de quels états ils peuvent être utilisés, lesquels sont utilisés, et les états destinataires. La figure IV.6 reprend elle l'utilisation faite par l'entité serveur des services session. Ce graphe est symétrique à celui décrit précédement.

Voyons maintenant comment nous avons résolu les problèmes plus pratiques. En ce qui concerne l'interface, le client utilise le type synchrone (il envoie une commande puis active la fonction de lecture d'événement synchrone). Pour le serveur, afin de résoudre simplement le problème de l'attente infinie des messages du client, nous avons aussi préféré utiliser la primitive de lecture synchrone qui dispose d'un "timer" d'inactivité. Pour résoudre le problème de l'adressage, nous avons ajouté à l'ISO DATABASE une nouvelle correspondance, celle entre le client, le fournisseur session, le serveur, et un identificateur d'application. Dans le texte du programme serveur, on recherche l'adresse du SSAP appelé (identificateur d'application) dans l'ISO DATABASE locale grâce au nom de l'entité client et du fournisseur session. L'ISO DATABASE appelée doit elle contenir la correspondance entre cette adresse et le nom du programme local (serveur) à activer. Seul donc l'identificateur d'application (adresse du SSAP appelé) doit être identique sur chaque système. Lorsque, sur un système, on fait appel au programme qui réalise le client, on lui fournit en plus du nom de fichier, le nom du système éloigné qui permettra de déterminer l'adresse réseau. Enfin, afin de pouvoir suivre le déroulement du transfert, les deux entités laissent (soit à l'écran, soit dans un fichier) une trace des événements d'interface (emploi ou réception d'une primitive de service) pour chacune des phases.

5 Evaluation personnelle

Notre évaluation porte principalement sur l'implémentation de la couche session. Nous apporterons toutefois quelques remarques sur la conception du logiciel en général.

La réalisation de la session n'est pas aussi complète que le manuel de l'utilisateur le laisse supposer : rappelons pour cela la non-réalisation des paramètres de qualité de session (à l'exception du contrôle étendu), d'un mécanisme de gestion des connexions transport (décision de garder une connexion transport, de la réutiliser par la suite), du service de signalisation d'anomalie par le fournisseur. Remarquons cependant que ces différents choix sont permis et simplifient la couche session.

Nous tenons à attirer l'attention sur le fait que, à notre avis, la session ISODE ne respecte pas tout à fait la norme ISO. En effet, de l'étude approfondie des programmes sources, nous avons constaté quelques irrégularités.

Nous n'avons pas eu le temps nécessaire pour nous attarder sur les performances de la session ISODE. Tout ce que nous pouvons en dire est que le C employé pour écrire les programmes est très élaboré : on y retrouve des conditions de compilation, toutes les catégories de définition de variables, un emploi continuel de pointeurs, des appels fréquents à des fonctions de UNIX. Ceci, quoique compliquant fortement l'approche des programmes par un néophyte, influence certainement de façon positive les performances de transfert et l'utilisation de la mémoire centrale.

Enfin, ce que nous avons beaucoup regretté, c'est le manque de documentation concernant la réalisation du logiciel : pas d'explication de l'architecture interne, commentaires inexistants dans les programmes, fichiers fantômes, ...

Les techniques d'interface, d'adressage et d'initialisation présentées pour la couche session sont à peu de choses près valables pour le reste des couches. Nous avons apprécié la simplicité avec laquelle les concepteurs ont réalisé l'interface, et la possibilité laissée aux utilisateurs de se servir de l'un ou l'autre type. De plus, l'architecture du logiciel, sa technique d'adressage et d'initialisation (quoique beaucoup plus complexes dans leurs réalisations que ne le laisse supposer notre présentation schématique), permettent de développer une couche de conception personnelle (un fournisseur session, une application reposant directement sur la session) et de l'installer facilement dans le "stack" ISODE.

Pour terminer, nous tenons à faire remarquer la cohérence de conception de l'ensemble des couches de service : cohérence dans les mécanismes généraux (interface, adressage, initialisation) et cohérence dans l'architecture des couches (l'architecture développée pour la session est applicable aux autres couches). Ceci facilite l'exploration de la réalisation du "stack" ISODE. Toutefois, le manque de documentation relaté pour la couche session reste d'actualité pour le reste des couches, et pour les autres programmes tels que ceux de test, de démonstration ...

6 Conclusion

Ce chapitre nous a permis de présenter une réalisation pratique des couches supérieures de l'ISO. Cette réalisation nous l'avions installée, "testée" et analysée. Cela nous a ouvert les yeux sur certains problèmes liés à l'expérience pratique, dont le plus difficile à résoudre s'est avéré être l'interface.

De notre étude d'ISODE, nous estimons que c'est une réalisation très riche (installable sur plusieurs OS, trois CASE disponibles, d'ici peu FTAM). De plus, si cette implémentation respecte le modèle de référence, elle s'ouvre aussi à la réalisation d'applications basées sur la norme X400.

En ce qui concerne la session, les concepteurs ont opté pour une réalisation simple (traitement d'erreur, gestion des connexions transport, paramètres de qualité de session), qui ne respecte pas tout à fait la norme ISO.

Retenons cependant les qualités essentielles de cette implémentation : la cohérence des mécanismes généraux, de l'architecture des couches, la possibilité laissée pour chaque couche de définir un des deux types d'interface, la possibilité de développer et d'installer facilement une couche ou une application de conception personnelle, et la qualité du C employé pour écrire l'ensemble des programmes.

V CONCLUSION

La couche session est un outil de synchronisation et d'organisation d'échange de données entre deux applications. Elle est par nature orientée connexion. Le travail de standardisation d'une session orientée non-connexion ne permettra en fait que de garder une portée générale au modèle de référence.

Les principaux concepts manipulés par ce niveau sont la structuration, le jeton, l'interruption et la reprise du dialogue, et la négociation. Les deux caractéristiques essentielles de la session sont les suivantes :

- programme à la carte et sous-ensemble
- mécanisme de structuration et d'organisation.

Grâce à son approche, la session ISO est extensible, suffisamment générale et spécialisable. De plus, à notre avis, cette approche ainsi que le niveau dans l'architecture des concepts concernés se justifient pleinement.

La réalisation, la mise en oeuvre des concepts dans les textes des différentes normes ont une influence sur les concepts et la qualité de généralité. Cette mise en oeuvre est assez complexe : ceci peut nuire à la compréhension et à une future utilisation des services de ce niveau. Remarquons toutefois la difficulté à laquelle ont dû s'affronter les concepteurs de cette couche qui travaillèrent de façon indépendante de la réalisation des autres niveaux : la session est utilisable par toute application, elle manipule des concepts abstraits et généraux; il n'est pas aisé d'imaginer tous les besoins en synchronisation et en organisation du dialogue, ainsi qu'une façon simple et efficace d'y répondre.

Nous avons insisté tout au long de notre travail sur le fait que le fournisseur session n'impliquait pas un overhead important sur les performances de transfert. De plus, les services offerts ne se limitent pas ainsi que pourrait le laisser supposer certains auteurs, aux trois sous-ensembles types. En fait, la session nous paraît un bon outil standard, assez riche même s'il ne couvre pas l'ensemble des applications utilisatrices.

La compréhension des services qui sont offerts et le choix d'un sous-ensemble particulier restent des problèmes assez complexes. L'analyse que nous avons faite permet de souligner les caractéristiques réelles de chacun des services. Quant au problème du

choix d'un sous-ensemble, la démarche que nous proposons permet de faciliter la tâche des futurs utilisateurs.

De notre maigre expérience pratique, nous pouvons opposer aux détracteurs de l'ISO qu'une implémentation des couches supérieures ça existe et ça tourne. ISODE en est un exemple assez riche. Toutefois, sa réalisation de la session est assez simple et ne permet pas de mieux comprendre des mécanismes tels que la concaténation étendue, la gestion des connexions transport, la réalisation de l'ensemble des paramètres de qualité de service.

Cette expérience permet quand même de montrer une réalisation de la couche session et une solution aux problèmes importants que sont l'interface, l'adressage et l'initialisation du stack ISO. De plus, elle nous a permis d'implémenter rapidement une petite application de transfert de fichiers illustrant nos propos. N'était-ce pas l'objectif premier des concepteurs d'ISODE ?

BIBLIOGRAPHIE

[CAN 86] Fausto Caneschi, "Hints for the interpretation of the ISO session layer", Computer Communication Review, vol. 16, no. 4, juillet / août 1986.

[CCR] ISO WD, "Information processing systems - Open systems interconnection - Definition of common application service elements for commitment, concurrency and recovery", ISO/TC97/SC16, novembre 1983.

[CHA 82] A. Lyman Chapin, "Connectionless Data Transmission", Computer Communication Review, vol. 12, no. 2, avril 1982.

[CHA 83] A. Lyman Chapin, "Connections and Connectionless Data Transmission", Proceedings of the IEEE, vol. 71, no. 12, décembre 1983.

[CUN 83] Ian Cunninghan, "Message-Handling Systems and protocols", Proceedings of th IEEE, vol 71, no. 12, décembre 1983

[DAY 83] John D. Day and Hubert Zimmermann, "The OSI Reference Model", Proceedings of the IEEE, vol. 71, no. 12, décembre 1983.

[EMM 83] Willard F. Emmons and A. S. Chandler, " OSI session layer : Services and Protocols", Proceedings of the IEEE, vol. 71, no. 12, décembre 1983.

[FTAMa] ISO DIS 8571/1, "Information processings systems - Open Systems Interconnection - File transfer, access and management - Part 1 : Introduction", ISO/TC97, 1987.

[FTAMb] ISO DIS 8571/1, "Information processings systems - Open Systems Interconnection - File transfer, access and management - Part 2 : the virtual filestore", ISO/TC97, 1987.

[ISO 7498] ISO IS 7498, " Systèmes de traitement de l'information - Interconnexion des systèmes ouverts - Modèle de référence de base", ISO/TC97/SC16, 1983.

[ISO 8326] ISO DIS 8326, "Systèmes de traitement de l'information - Interconnexion des systèmes ouverts - Définition du service session de base en mode connexion", ISO/TC97/SC16, 1983.

[ISO 8327] ISO DIS 8327, " Systèmes de traitement de l'information - Interconnexion des systèmes ouverts - Définition du protocole session de base en mode connexion", ISO/TC97/SC16, 1983.

[LAN 83] Alwyn Langford, Kenji Naemura, and R. Speth, "OSI Management and Job Transfer Service", Proceedings of the IEEE, vol. 71, no. 12, décembre 1983.

[LEW 83] Douglas Lewan and H. Garret Long, "The OSI File Service", Proceedings of the IEEE, vol. 71, no. 12, décembre 1983.

[LOW 83] Henry Lowe, "OSI Virtual Terminal Service", Proceedings of the IEEE, vol. 71, no.12, décembre 1983.

[OMN 86] OMNICOM Information Service, "Open Systems Data Transfer", février 1986.

[POP 84] R. Popescu-Zeletin, "Some critical considerations on the ISO/OSI RM from a network implementation point of vue", Computer Communication Review, vol. 14, no. 12, 1984.

[RAU 86] Wendy Rauch-Hindin, " Upper level OSI protocols near completion", Mini-Micro Systems, juillet 1986.

[ROS 87] Marshall T. Rose and Dwight E. Cass, "The ISO Development Environment at NRTC : User's Manual", mai 1987.

[T 62] CCITT, "Control procedures for teletex and group 4 facsimile services", 1984.

[X 410] CCITT, "Message Handling Systems : Remote Operations and Reliable Transfer Server", 1984.

ANNEXE

SERVEUR

réception indication de connexion		
acceptation de la demande		
TANT QUE pas-terminé		
attend message		
CAS message	coupure fournis	FIN.
	message inconnu	RESYNCHRO
	données typées	TRAIT-TYP
	AUTRES	TRAIT-EVEN
TANT QUE pas-terminé		
attend message		
CAS message	coupure utilisat.	FIN
	message inconnu	RESYNCHRO
	données	écriture page
	synchronisation	SYNCINDICATION
	terminais. connex.	acceptation
	AUTRES	FIN

FIN : arrêt du programme

OK : continuation

BREAK : sortie de l'ensemble des structures imbriquées

TRAIT-TYP

<table>
<tr><td colspan="4">vérification de l'existence du fichier</td></tr>
<tr><td colspan="3">OUI — fichier existe</td><td>NON</td></tr>
<tr><td colspan="3">envoi résultat</td><td>envoi résultat</td></tr>
<tr><td colspan="3">attend message</td><td rowspan="5">BREAK</td></tr>
<tr><td rowspan="4">CAS message</td><td rowspan="2">termin. connex.</td><td>confirmation</td></tr>
<tr><td>FIN</td></tr>
<tr><td rowspan="2">AUTRES</td><td>coupure utilis.</td></tr>
<tr><td>FIN</td></tr>
</table>

RESYNCHRO

<table>
<tr><td colspan="3">demande de resynchronisation</td></tr>
<tr><td colspan="3">attend réponse</td></tr>
<tr><td rowspan="2">CAS mes</td><td>conf. resync.</td><td>OK</td></tr>
<tr><td>AUTRES</td><td>FIN</td></tr>
</table>

TRAIT-EVEN

OUI / resynchronisation	NON
confirmation	coupure utilisateur

SYNCINDICATION

CAS synchr	indic. mineur	réponse mineur
	indic. resync.	réponse resynchronisation
	AUTRES	coupure utilisateur

CLIENT

demande de connexion

réponse = refus

OUI | NON

FIN | OK

TANT QUE pas-terminé

envoi du nom du fichier

attend réponse

CAS réponse		
	coupure fournis.	FIN
	données typées	fichier existe : OUI → DECONNEXION ; NON → BREAK
	message inconnu	RESYNCHRO1
	AUTRES	TRAIT-EVEN

TANT QUE il-y-a-des-pages

envoi page

envoi mineur

attend réponse

CAS réponse		
	coupure fournis.	FIN
	confirmat. mineur	OK
	indic. resynchron.	réponse et reprise (envoi page)
	message inconnu	RESYNCHRO 2
	AUTRES	FIN

DECONNEXION

DECONNEXION

demande de déconnexion	
OUI — réponse = refus	NON
coupure utilisateur	FIN
FIN	

TRAIT-EVEN

CAS evenem	jetons	OK
	indic. resync.	réponse resynchronisation
	AUTRES	FIN

RESYNCHRO

demande de resynchronisation		
attend réponse		
CAS mes	conf. resync.	envoi page précédente (pour 2, sinon BEAK)
	AUTRES	FIN

```
/*                          SERVEUR                          */

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <isode/ssap.h>
#include <isode/isoservent.h>

#define DEBUG

void   adios (), advise (), ss_adios (), ss_advise ();
int ss_dataindication (), ss_syncindication (), ss_finishindication ();

static char *myname = "srfiles ";

static int owned = 0;
static int avail = 0;
static int secs  = 15;

static long ssn;

char   buffer [BUFSIZ];
FILE *fopen (), *fp, *ferr;

/*
```

```
 */

main (argc, argv, envp)
int     argc;
char  **argv,
      **envp;
{
    int    result,
           tokens,
           sd,
           requirements,
           i;
    struct SSAPstart                sss;
    register struct SSAPstart      *ss = &sss;
    struct SSAPindication           sis;
    register struct SSAPindication *si = &sis;
    struct SSAPdata                 sxs;
    register struct SSAPdata       *sx = &sxs;
    register struct SSAPabort      *sa = &si -> si_abort;
    register struct SSAPsync       *sn = &si -> si_sync;
    struct stat                     buf;

    ferr = fopen ("/usr/local/src/isode-2.0/support/iso.srf.err","w");
    advise (NULLCP,"C EST PARTI MON ...");

    if (SInit (argc,argv,ss,si) == NOTOK )
        ss_adios (sa, "initialization fails");

    advise (NULLCP,"INIT TERMINE");

    sd = ss -> ss_sd;
    requirements = ss -> ss_requirements;
    tokens = ss -> ss_settings;

    if (SConnResponse (sd, &ss -> ss_connect, NULLSA,SC_ACCEPT,
                       ss -> ss_requirements,ss -> ss_settings,
                       ss -> ss_isn,NULLCP,0,si) == NOTOK)
        ss_adios (sa,"S-CONNECT.RESPONSE (accept)");

    advise (NULLCP,"CONNECT TERMINE");
/*
```

```
*/
    for (;;) {

      switch (result = SReadRequest (sd,sx,secs,si)) {

          case DONE  :

              if (   (si -> si_type == SI_SYNC)
                  && (sn -> sn_type == SN_RESETIND) ) {
                advise (NULLCP,"RESYNC ARRIVED");
                if (SReSyncResponse (sd,sn -> sn_ssn,NULLCP,0,si) == NOTOK)
                   ss_adios (sa,"S-RESYNC.RESPONSE");
                }
              else {
                (void) strcpy (buffer,"protocol mismatch");
                SUAbortRequest (sd, buffer,strlen (buffer) + 1,si);
                adios (NULLCP,"protocol mismatch");
               }
               continue;

          case NOTOK :
              ss_adios (sa,"S-ABORT.INDICATION");

          case  OK   :
     {
      advise (NULLCP,"NOM FICHIER RECU");

      if (stat (sx -> sx_base,&buf) == 0) {

           advise (NULLCP,"FICHIER EXISTANT : ENVOI TYPED");

           strcpy(buffer,"NOTOK");
           if (STypedRequest (sd,buffer,strlen (buffer) + 1,
                              si) == NOTOK)
               ss_adios (sa,"S-TYPED-DATA.REQUEST");

           advise (NULLCP,"REPONSE ENVOYEE");

           SXFREE (sx);
           switch (result = SReadRequest(sd,sx,secs,si)) {
               case NOTOK :
                  ss_adios (sa,"S-READ.REQUEST");
               case  OK   :
               case DONE  :
                   if (si -> si_type == SI_FINISH) {
                     if (SRelResponse (sd,SC_ACCEPT,NULLCP,0,si) == NOTOK)
                       ss_adios (sa,"S-RELEASED.>RESPONSE");
                       adios (NULLCP,"connection released");
                     }
                   else {
                     (void) strcpy (buffer,"protocol mismatch");
                     SUAbortRequest (sd, buffer,strlen (buffer) + 1,si);
                     adios (NULLCP,"protocol mismatch");
                    }
                default  :
                     adios (NULLCP,"unrecognized PDU");

           }
       }

/*
```

```
*/

      advise (NULLCP,"ENVOI TYPED");

      (void) strcpy (buffer,"ok");
      if (STypedRequest (sd,buffer,strlen (buffer) + 1,si) == NOTOK)
        ss_adios (sa,"S-TYPED-DATA.REQUEST");

      fp = fopen ( sx -> sx_base,"w");
      advise (NULLCP,"NOM DU FICHIER : %s",sx -> sx_base);
     }
      break;

            default :
                 advise (NULLCP," unknown  indication");
                 if (SReSyncRequest (sd,SYNC_SET,ssn,tokens,NULLCP,0,si)
                        == NOTOK )
                        ss_adios (sa,"S-RESYNCHRONIZE.REQUEST");
                 switch (result = SReadRequest (sd,sx,secs,si) == NOTOK ) {
                        case DONE  :
                                advise (NULLCP,"resync confirmation");
                                break;
                        default    :
                                ss_adios (sa,"S-READ.REQUEST");
                 }
                 continue;
           }
    break;
    }

/*
```

```
*/

    advise (NULLCP,"BOUCLE DE RECEPTION");


    for (i = 1;;i++) {
        advise (NULLCP,"ITERATION : %d ",i);
        switch (result = SReadRequest (sd,sx,secs,si)) {
            case NOTOK :
                ss_adios (sa,"S-READ.REQUEST");
            case  OK   :
                ss_dataindication (sd,sx);
                break;
            case DONE  :
                switch (si -> si_type) {
                    case SI_SYNC :
                        ss_syncindication(sd,&si -> si_sync);
                        break;
                    case SI_FINISH :
                        ss_finishindication (sd,&si -> si_finish);
                    default :
                        adios (NULLCP,"unknown indication");
                }
                break;
            default :
                advise (NULLCP,"unknown indication");
                if (SReSyncRequest (sd,SYNC_SET,ssn,tokens,NULLCP,0,si)
                        == NOTOK )
                        ss_adios (sa,"S-RESYNCHRONIZE.REQUEST");
                (void) strcpy (buffer, "PROBLEME LIGNE PRECEDENTE");
                fputs (buffer,fp);
                switch (result = SReadRequest (sd,sx,secs,si) == NOTOK ) {
                        case DONE  :
                                advise (NULLCP,"resync confirmation");
                                break;
                        default    :
                                ss_adios (sa,"S-READ.REQUEST");
                }
        }
    }
}
/*
```

```
*/
    static int ss_dataindication (sd,sx)
    int sd;
    register struct SSAPdata *sx;
    {
        advise (NULLCP,"DATA ARRIVED");

        fputs (sx -> sx_base,fp);
        SXFREE (sx);

    }


    static int ss_syncindication (sd,sn)
    int sd;
    register struct SSAPsync *sn;
    {
        struct SSAPindication            sis;
        register struct SSAPindication *si = &sis;
        register struct SSAPabort      *sa = &si -> si_abort;

        switch (sn -> sn_type) {
            case SN_MINORIND :

                advise (NULLCP,"SYNC ARRIVED");

                if (SMinSyncResponse (sd,sn -> sn_ssn,NULLCP,0,si) == NOTOK)
                   ss_adios (sa,"S-MINOR.RESPONSE");
                ssn = sn -> sn_ssn;
                break;
            case SN_RESETIND :

                advise (NULLCP,"RESYNC ARRIVED");

                 if (SReSyncResponse (sd,sn -> sn_ssn,NULLCP,0,si) == NOTOK)
                   ss_adios (sa,"S-RESYNC.RESPONSE");
                 (void) strcpy (buffer, "PROBLEME LIGNE PRECEDENTE");
                 fputs (buffer,fp);
                 break;
            default :
                   (void) strcpy (buffer,"protocol mismatch");
                   SUAbortRequest (sd, buffer,strlen (buffer) + 1,si);
                   adios (NULLCP,"protocol mismatch");
            }
    }

/*
```

```
*/

    static int ss_finishindication (sd,sf)
    int sd;
    register struct SSAPfinish *sf;
    {
        struct SSAPindication           sis;
        register struct SSAPindication *si = &sis;
        register struct SSAPabort      *sa = &si -> si_abort;

        advise (NULLCP,"S-RELEASE.INDICATION");
        if (SRelResponse (sd,SC_ACCEPT,NULLCP,0,si) == NOTOK )
          ss_adios (sa,"S-RELEASE.INDICATION");
        fclose (fp);
        fclose (ferr);
        exit (0);
     }
/*
```

```
 */

static void  ss_adios (sa, event)
register struct SSAPabort *sa;
char   *event;
{
    ss_advise (sa, event);

    fclose (ferr);
    fclose (fp);
    _exit (1);
}


static void  ss_advise (sa, event)
register struct SSAPabort *sa;
char   *event;
{
    char    buffer[BUFSIZ];

    if (sa -> sa_cc > 0)
        (void) sprintf (buffer, "[%s] %*.*s",
                SErrString (sa -> sa_reason),
                sa -> sa_cc, sa -> sa_cc, sa -> sa_data);
    else
        (void) sprintf (buffer, "[%s]", SErrString (sa -> sa_reason));

    advise (NULLCP, "%s: %s%s", event, buffer,
            sa -> sa_peer ? " (peer Initiated)" : "");
}



static void  adios (what, fmt, a, b, c, d, e, f, g, h, i, j)
char   *what,
       *fmt,
       *a,
       *b,
       *c,
       *d,
       *e,
       *f,
       *g,
       *h,
       *i,
       *j;
{
    advise (what, fmt, a, b, c, d, e, f, g, h, i, j);
    fclose (ferr);
    fclose (fp);
    _exit (1);
}

/*
```

```
 */

/* VARARGS3 */

void  advise (what, fmt, a, b, c, d, e, f, g, h, i, j)
char   *what,
       *fmt,
       *a,
       *b,
       *c,
       *d,
       *e,
       *f,
       *g,
       *h,
       *i,
       *j;
{
    (void) fflush (stdout);

    fprintf (ferr, "%s: ", myname);
    fprintf (ferr, fmt, a, b, c, d, e, f, g, h, i, j);
    if (what)
        (void) fputc (' ', ferr), perror (what);
    else
        (void) fputc ('\n', ferr);
    (void) fflush (ferr);
}
```

```
/*                              CLIENT                                  */

#include <stdio.h>
#include <isode/ssap.h>
#include <isode/isoservent.h>


#define DEBUG


void    adios (), advise (), ss_adios (), ss_advise ();

static char *myname = "ssfilec ";
static char *notok  = "NOTOK";

static int requirements = SR_HALFDUPLEX
                | SR_MINORSYNC | SR_RESYNC
                | SR_TYPEDATA;

static int owned = 0;
static int avail = 0;

static long ssn;

/*
```

```
 */

main (argc, argv, envp)
int     argc;
char  **argv,
      **envp;
{
    register struct isoservent *is;
    int     sd,
            cc,
            i,
            j,
            k,
            l,
            tokens;
    char   *cp,
           *dp,
            buffer[BUFSIZ],
            page[2000],
           *fpagesend();
    FILE *fopen (), *fp;
    register struct SSAPaddr *sz;
    struct SSAPref  sfs;
    register struct SSAPref *sf;
    struct SSAPconnect  scs;
    register struct SSAPconnect *sc = &scs;
    struct SSAPrelease  srs;
    register struct SSAPrelease *sr = &srs;
    struct SSAPindication   sis;
    register struct SSAPindication *si = &sis;
    register struct SSAPabort  *sa = &si -> si_abort;

    int     result;
    struct SSAPdata sxs;
    register struct SSAPdata   *sx = &sxs;

    register struct SSAPsync   *sn = &si -> si_sync;


/*
```

```
*/

    advise (NULLCP,"C EST PARTI MON ...");

    if ((is = getisoserventbyname ("ssfilec","ssap")) == NULL)
        adios (NULLCP, "ssap/sink: unknowm ISO provider/entity pair");
    if ((sz = is2saddr (argv[1], is)) == NULL)
        adios (NULLCP, "address translation failed");
    advise (NULLCP,"address : %s  %d",sz -> sa_selector,sz -> sa_selectlen);
    if ((sf = addr2ref (SLocalHostName ())) == NULL) {
        sf = &sfs;
        (void) bzero ((char *) sf, sizeof *sf);
    }

    tokens = 0;
    ssn = 0;
#define dotoken(requires,shift,bit,type) \
{ \
    if (requirements & requires) \
        tokens |= ST_INIT_VALUE << shift; \
}
    dotokens ();
#undef  dotoken


    fprintf (stderr, "%s... ", argv[1]);
    if (SConnRequest (sf, NULLSA, sz, requirements, tokens,
         ssn,NULLCP,0, sc, si) == NOTOK) {
        fprintf (stderr, "failed\n");
        ss_adios (sa, "S-CONNECT.REQUEST");
    }

    if (sc -> sc_result != SC_ACCEPT) {
        fprintf (stderr, "failed\n");
        if (sc -> sc_cc > 0)
            adios (NULLCP, "Connection rejected: [%s] %*.*s",
                        SErrString (sc -> sc_result),
                        sc -> sc_cc, sc -> sc_cc, sc -> sc_data);
        else
            adios (NULLCP, "Connection rejected: [%s]",
                        SErrString (sc -> sc_result));
    }


    fprintf (stderr, "connected\n");
    fprintf (stderr,"reference : \n u :%s \n  u2 :%s \n  a :%s  v:%s \n",
             sf -> sr_udata,sf -> sr_cdata,sf -> sr_adata,sf -> sr_vdata);

    sd = sc -> sc_sd;

    requirements = sc -> sc_requirements;

/*
```

```
 */

#define dotoken(requires,shift,bit,type) \
{ \
    if (requirements & requires) \
        switch (sc -> sc_settings & (ST_MASK << shift)) { \
            case ST_CALL_VALUE: \
                adios (NULLCP, "%s token: choice", type); \
 \
            case ST_INIT_VALUE: \
                owned |= bit, avail |= bit; \
                break; \
 \
            case ST_RESP_VALUE: \
                avail |= bit; \
                break; \
 \
            default: \
                adios (NULLCP, "%s token: reserved", type); \
        } \
}
        dotokens ();
#undef  dotoken

/*
```

```
*/

for (;;) {
        advise (NULLCP,"ENVOI DU NOM DE FICHIER");

        strcpy (buffer,argv[3]);
        if (argc != 4)
            for (i = 4;i < argc;i++)
                strcat(buffer,argv[i]);
        if (SDataRequest(sd,buffer,sizeof buffer,si) == NOTOK)
            ss_adios (sa,"S-DATA-REQUEST");

        advise (NULLCP,"ENVOI DU NOM DE FICHIER EFFECTUE");


    switch(result = SReadRequest(sd,sx,NOTOK,si)) {
         case NOTOK :
             ss_adios (sa,"S-READ.REQUEST");
         case  OK   :
             advise (NULLCP,"EVENEMENT ENTRANT : REPONSE ARRIVEE");
             if (strcmp(sx -> sx_base,notok) == 0) {
                advise (NULLCP," file already exists on remotehost ");
                SXFREE (sx);

                if (SRelRequest (sd, NULLCP, 0, sr, si) == NOTOK)
                   ss_adios (sa, "S-RELEASE.REQUEST");

                advise (NULLCP,"RELEASE ENVOYEE");

                if (!sr -> sr_affirmative) {
                   (void) SUAbortRequest (sd, NULLCP, 0, si);
                    if (sr -> sr_cc > 0)
                      adios (NULLCP, "Release rejected by peer: %*.*s",
                          sr -> sr_cc, sr -> sr_cc, sr -> sr_data);
                    else
                      adios (NULLCP, "Release rejected by peer");
                 }
                 exit (0);
             }
             break;
         case DONE  :
             advise (NULLCP,"EVENEMENT ENTRANT");
             ss_event (sd,si);
             continue;
         default    :
             advise (NULLCP," unknown return from SReadRequest ",result);
             if (SReSyncRequest (sd,SYNC_SET,ssn,tokens,NULLCP,0,si)
                == NOTOK)
                         ss_adios (sa,"S-RESYNCHRONIZE.REQUEST");
             switch (result = SReadRequest (sd,sx,NOTOK,si)) {
                case DONE  :
                         advise (NULLCP,"resync confirmation");
                         break;
                default    :
                         adios (NULLCP,"protocol mismatch");
             }
             break;
     }
     break;
}

SXFREE (sx);
/*
```

```
*/

advise (NULLCP,"DEBUT ENVOI FICHIER");

fp = fopen(argv[2],"r");
while (fpagesend(page,fp)) {
    if (SDataRequest(sd,page,sizeof page,si) == NOTOK)
        ss_adios (sa,"S-DATA.REQUEST");
    advise (NULLCP,"ENVOI PAGE");
    if (SMinSyncRequest (sd,SYNC_CONFIRM,&ssn,NULLCP,0,si) == NOTOK)
        ss_adios (sa,"S-MINOR-SYNC.REQUEST");
    advise (NULLCP,"ENVOI MINOR");
    switch(result = SReadRequest(sd,sx,NOTOK,si)) {
        case  OK   :
        case NOTOK :
            ss_adios (sa,"S-READ.REQUEST");
        case DONE  :
            if((si -> si_type == SI_SYNC) && (sn -> sn_type == SN_MINORCNF):
                break;
            if((si -> si_type == SI_SYNC) && (sn -> sn_type == SN_RESETIND):
              { advise (NULLCP, "ENVOI REPONSE RESYNC");
                if (SReSyncResponse (sd,sn -> sn_ssn,tokens,NULLCP,0,si)
                    == NOTOK)
                     ss_adios (sa,"S-RESYNMCHRONIZE.RESPONSE");
                advise (NULLCP, "resync confirmation");
                if (SDataRequest(sd,buffer,sizeof buffer,si) == NOTOK)
                     ss_adios (sa,"S-DATA.REQUEST");
                break; }
            else  adios (NULLCP," protocol mismatch ",result);

        default    :

             { advise (NULLCP, " unknown return from SReadRequest ");
               if (SReSyncRequest (sd,SYNC_SET,ssn-1,tokens,NULLCP,0,si)
                   == NOTOK)
                    ss_adios (sa,"S-RESYNCHRONIZE.REQUEST");
               switch (result = SReadRequest (sd,sx,NOTOK,si)) {
                       case DONE :
                         advise (NULLCP, "resync confirmation");
                         if (SDataRequest(sd,buffer,sizeof buffer,si) == NOTO
                               ss_adios (sa,"S-DATA.REQUEST");
                         break;
                       case OK :
                       case NOTOK :
                       default :
                            ss_adios (sa,"S-READ.REQUEST");
                   }
               break;
             }
    }
}

/*
```

```
*/

    fclose (fp);

    if (SRelRequest (sd, NULLCP, 0, sr, si) == NOTOK)
        ss_adios (sa, "S-RELEASE.REQUEST");

    if (!sr -> sr_affirmative) {
        (void) SUAbortRequest (sd, NULLCP, 0, si);
        if (sr -> sr_cc > 0)
            adios (NULLCP, "Release rejected by peer: %*.*s",
                sr -> sr_cc, sr -> sr_cc, sr -> sr_data);
         else
            adios (NULLCP, "Release rejected by peer");
        }

exit (0);
}

/*
```

```
*/

static  ss_event (sd, si)
int     sd;
register struct SSAPindication *si;
{
    register struct SSAPabort  *sa = &si -> si_abort;
    register struct SSAPactivity  *sv = &si -> si_activity;
#ifdef  DEBUG
    register struct SSAPreport  *sp = &si -> si_report;
#endif
    register struct SSAPsync   *sn = &si -> si_sync;
    register struct SSAPtoken  *st = &si -> si_token;

    switch (si -> si_type) {
        case SI_TOKEN:
            advise (NULLCP,"TOKEN ARRIVED");
            switch (st -> st_type) {
                case ST_GIVE:
                case ST_CONTROL:
                    owned = st -> st_owned;
                    break;

                case ST_PLEASE:
                    break;

                default:
                    adios (NULLCP,
                            "unknown token indication type=0x%x, %d bytes",
                            st -> st_type, st -> st_cc);
            }
            break;

            case SI_SYNC :
              switch (sn -> sn_type) {
                case SN_RESETIND:
#ifdef  DEBUG
                    advise (NULLCP, "resync indication type=%d %d, %d bytes"
                            sn -> sn_options, sn -> sn_ssn, sn -> sn_cc);
#endif
/*
```

```
#define dotoken(requires,shift,bit,type) \
{ \
                    if (requirements & requires) \
                        switch (sn -> sn_settings & (ST_MASK << shift)) { \
                            case ST_CALL_VALUE << shift: \
                                sn -> sn_settings &= ~(ST_MASK << shift); \
                                sn -> sn_settings |= ST_RESP_VALUE << shift
                            case ST_RESP_VALUE << shift: \
                                owned &= ~bit; \
                                break; \
 \
                            case ST_INIT_VALUE << shift: \
                                owned |= bit; \
                                break; \
 \
                            default: \
                                adios (NULLCP, "%s token: reserved", type); \
                                break; \
                        } \
}
                    dotokens ();
#undef  dotoken
                    advise (NULLCP, "ENVOI REPONSE RESYNC");
                    if (SReSyncResponse (sd, sn -> sn_ssn, sn -> sn_settings
                                NULLCP, 0, si) == NOTOK)
                        ss_adios (sa, "S-RESYNCHRONIZE.RESPONSE");
                    break;

                case SN_RESETCNF:
#ifdef  DEBUG
                    advise (NULLCP, "resync confirmation %d, %d bytes",
                            sn -> sn_ssn, sn -> sn_cc);
#endif
                    break;

                default:
                    adios (NULLCP,
                            "unknown sync indication=0x%x, ssn=%d, %d bytes"
                            sn -> sn_type, sn -> sn_ssn, sn -> sn_cc);
            }
            break;

        default :
                adios (NULLCP," unknown indication type = 0x%x", si -> si_typ
    }
}

/*
```

```
static void  ss_adios (sa, event)
register struct SSAPabort *sa;
char   *event;
{
    ss_advise (sa, event);

    _exit (1);
}


static void  ss_advise (sa, event)
register struct SSAPabort *sa;
char   *event;
{
    char    buffer[BUFSIZ];

    if (sa -> sa_cc > 0)
        (void) sprintf (buffer, "[%s] %*.*s",
                SErrString (sa -> sa_reason),
                sa -> sa_cc, sa -> sa_cc, sa -> sa_data);
    else
        (void) sprintf (buffer, "[%s]", SErrString (sa -> sa_reason));

    advise (NULLCP, "%s: %s%s", event, buffer,
            sa -> sa_peer ? " (peer Initiated)" : "");
}



static void  adios (what, fmt, a, b, c, d, e, f, g, h, i, j)
char   *what,
       *fmt,
       *a,
       *b,
       *c,
       *d,
       *e,
       *f,
       *g,
       *h,
       *i,
       *j;
{
    advise (what, fmt, a, b, c, d, e, f, g, h, i, j);
    _exit (1);
}

/*
```

```
..

/* VARARGS3 */

void  advise (what, fmt, a, b, c, d, e, f, g, h, i, j)
char   *what,
       *fmt,
       *a,
       *b,
       *c,
       *d,
       *e,
       *f,
       *g,
       *h,
       *i,
       *j;
{
    (void) fflush (stdout);

    fprintf (stderr, "%s: ", myname);
    fprintf (stderr, fmt, a, b, c, d, e, f, g, h, i, j);
    if (what)
        (void) fputc (' ', stderr), perror (what);
    else
        (void) fputc ('\n', stderr);
    (void) fflush (stderr);
}


char *fpagesend (page,fp)
char *page;
FILE *fp;
{
    register int i;
    char         buffer[2000];

    *page = '\0';
    for (i=1;(i<=24) && (fgets(buffer,sizeof buffer,fp));i++)
         strcat (page,buffer);
    if (*page == '\0') advise (NULLCP,"FIN FICHIER");
    return ((*page == '\0') ? NULL : page);
 }
```